Jdon 框架使用开发指南

板桥里人(banq J道 http://www.jdon.com)

1.2.2 版本： 2005 年 7 月 14 日

JdonFramework下载地址：http://sourceforge.net/projects/jdon/

技术支持论坛：http://www.jdon.com/jive/forum.jsp?forum=61&thRange=30

技术背景

J2EE 基本概念

J2EE 可以说指 Java 在数据库信息系统上实现，数据库信息系统从早期的 dBase、到 Delphi/VB 等 C/S 结构，发展到 B/S（Browser 浏览器/Server 服务器）结构，而 J2EE 主要是指 B/S 结构的实现。

J2EE 又是一种框架和标准，框架类似 API、库的概念，但是要超出它们。

J2EE 是一个虚的大的概念，J2EE 标准主要有三种子技术标准：WEB 技术、EJB 技术和 JMS，谈到 J2EE 应该说最终要落实到这三个子概念上。

这三种技术的每个技术在应用时都涉及两个部分：容器部分和应用部分，Web 容器也是指 Jsp/Servlet 容器，你如果要开发一个 Web 应用，无论是编译或运行，都必须要有 Jsp/Servlet 库或 API 支持（除了 JDK/J2SE 以外）。

Web 技术中除了 Jsp/Servlet 技术外，还需要 JavaBeans 或 Java Class 实现一些功能或者包装携带数据，所以 Web 技术最初裸体简称为 Jsp/Servlet+JavaBeans 系统。

谈到 JavaBeans 技术，就涉及到组件构件技术（component），这是 Java 的核心基础部分，很多软件设计概念（设计模式）都是通过 JavaBeans 实现的。

JavaBeans 不属于 J2EE 概念范畴中，如果一个 JavaBeans 对象被 Web 技术（也就是 Jsp/Servlet）调用，那么 JavaBeans 就运行在 J2EE 的 Web 容器中；如果它被 EJB 调用，它就运行在 EJB 容器中。

EJB（企业 JavaBeans）是普通 JavaBeans 的一种提升和规范，因为企业信息系统开发中需要一个可伸缩的性能和事务、安全机制，这样能保证企业系统平滑发展，而不是发展到一种规模重新更换一套软件系统。

至此，JavaBeans 组件发展到 EJB 后，并不是说以前的那种 JavaBeans 形式就消失了，这就自然形成了两种 JavaBeans 技术：EJB 和 POJO，POJO 完全不同于 EJB 概念，指的是普通 JavaBeans，而且这个 JavaBeans 不依附某种框架，或者干脆可以说：这个 JavaBeans 是你为这个应用程序单独开发创建的。

J2EE 应用系统开发工具有很多：如 JBuilder、Eclipse 等，这些 IDE 首先是 Java 开发工具，也就是说，它们首要基本功能是可以开发出 JavaBeans 或 Java class，但是如果要开发出 J2EE 系统，就要落实到要么是 Web 技术或 EJB 技术，那么就有可能要一些专门模块功能，最重要的是，因为 J2EE 系统区分为容器和应用两个部分，所以，在任何开发工具中开发 J2EE 都需要指定 J2EE 容器。

J2EE 容器分为 WEB 容器和 EJB 容器，Tomcat/Resin 是 Web 容器；JBoss 是 EJB 容器+Web 容器等，其中 Web 容器直接使用 Tomcat 实现的。所以你开发的 Web 应用程序可以在上面两种容器运行，而你开发的 Web+EJB 应用则只可以在 JBoss 服务器上运行，商业产品 Websphere/Weblogic 等和 JBoss 属于同一种性质。

J2EE 容器也称为 J2EE 服务器，大部分时它们概念是一致的。

如果你的 J2EE 应用系统的数据库连接是通过 JNDI 获得，也就是说是从容器中获得，那么你的 J2EE 应用系统基本与数据库无关，如果你在你的 J2EE 应用系统耦合了数据库 JDBC 驱动的配置，那么你的 J2EE 应用系统就有数据库概念色彩，作为一个成熟需要推广的 J2EE 应用系统，不推荐和具体数据库耦合，当然这其中如何保证 J2EE 应用系统运行性能又是体现你的设计水平了。

如何开发一个高质量的 J2EE 系统

衡量 J2EE 应用系统设计开发水平高低的标准就是：解耦性；你的应用系统各个功能是否能够彻底脱离？是否不相互依赖，也只有这样，才能体现可维护性、可拓展性的软件设计目标。

为了达到这个目的，诞生各种框架概念，J2EE 框架标准将一个系统划分为 WEB 和 EJB 主要部分，当然我们有时不是以这个具体技术区分，而是从设计上抽象为表现层、服务层和持久层，这三个层次从一个高度将 J2EE 分离开来，实现解耦目的。

因此，我们实际编程中，也要将自己的功能向这三个层次上靠，做到大方向清楚，泾渭分明，但是没有技术上约束限制要做到这点是很不容易的，因此我们还是必须借助 J2EE 具体技术来实现，这时，你可以使用 EJB 规范实现服务层和持久层，Web 技术实现表现层；

EJB 为什么能将服务层从 Jsp/Servlet 手中分离出来，因为它对 JavaBeans 编码有强制的约束，现在有一种对 JavaBeans 弱约束，使用 Ioc 模式实现的（当然 EJB 3.0 也采取这种方式），在 Ioc 模式诞生前，一般都是通过工厂模式来对 JavaBeans 约束，形成一个服务层，这也是是 Jive 这样开源论坛设计原理之一。

由此，将服务层从表现层中分离出来目前有两种可选架构选择：管理普通 JavaBeans（POJO）框架(如 Spring、JdonFramework)以及管理 EJB 的 EJB 框架，因为 EJB 不只是框架，还是标准，而标准可以扩展发展，所以，这两种区别将来是可能模糊，被纳入同一个标准了。

但是，通常标准制定是为某个目的服务的，总要牺牲一些换取另外一些，所以，这两种架构会长时间并存。

前面谈了服务层框架，使用服务层框架可以将 JavaBeans 从 Jsp/Servlet 中分离出来，而使用表现层框架则可以将 Jsp 中剩余的 JavaBeans 完全分离，这部分 JavaBeans 主要负责显示相关，一般是通过标签库（taglib）实现，不同框架有不同自己的标签库，Struts 是应用比较广泛的一种表现层框架。

这样，表现层和服务层的分离是通过两种框架达到目的，剩余的就是持久层框架了，通过持久层的框架将数据库存储从服务层中分离出来是其目的，持久层框架有两种方向：直接自己编写 JDBC 等 SQL 语句（如 iBatis）；使用 O/R Mapping 技术实现的 Hibernate 和 JDO 技术；当然还有 EJB 中的实体 Bean 技术。

持久层框架目前呈现百花齐放，各有优缺点的现状，所以正如表现层框架一样，目前没有一个框架被指定为标准框架，当然，表现层框架现在又出来了一个 JSF，它代表的页面组件概念是一个新的发展方向，但是复杂的实现让人有些忘而却步。

最后，你的 J2EE 应用系统如果采取上面提到的表现层、服务层和持久层的框架实现，基本可以在无需深刻掌握设计模式的情况下开发出一个高质量的应用系统了。

还要注意的是: 开发出一个高质量的 J2EE 系统还需要正确的业务需求理解，那么域建模提供了一种比较切实可行的正确理解业务需求的方法，相关详细知识可从 UML 角度结合理解。

当然，如果你想设计自己的行业框架，那么第一步从设计模式开始吧，因为设计模式提供你一个实现 JavaBeans 或类之间解耦参考实现方法，当你学会了系统基本单元 JavaBeans 或类之间解耦时，那么系统模块之间的解耦你就可能掌握，进而你就可以实现行业框架的提炼了，这又是另外一个发展方向了。

以上理念可以总结为一句话：

J2EE 开发三件宝: Domain Model（域建模）、patterns（模式）和 framework（框架）。

域驱动开发框架

Ruby on Rails 已经受到越来越多的重视，更多文章开始关注，Rolling with Ruby on Rails 一文比较详细，也有国人做了翻译，按这里。这里我们不过多讨论 ROR(Ruby on Rails)，而是探讨如何以一个正确的方式快速开发 J2EE。

现在的问题

现在我们 J2EE 开发碰到了什么问题呢？ 让我们想象一下使用 Spring 和 Hibernate 开发一个 J2EE WEB 应用是什么样的：我们需要增加一个新的域对象类型为 Person，下面主要的开发步骤：

1.创建 Person 类.
2.创建 PersonDAO 类.
3.创建 Person 数据表.
4.定义 PersonDAO 在 Spring 的 application context XML 文件.
5.创建 Person page 页面和 action 类.
6.增加 Person 页面到 web 框架(如 struts)XML 配置文件中.
7.创建 personList 页面来显示 Person 实例.
8.创建 personEdit 页面来编辑 Person 实例.
你会确实感慨：真是需要很多步骤啊。

如何解决?

关键问题是我们开发时不能重复一些步骤，因此必须尽量减少步骤，如果只减少步骤到:
1.创建 Person 类?
是否只需要第一个步骤就可以？在第一步时，我们花费更多时间精力进行域建模，确定域模型的属性行为等。其他步骤我们会发现下面的规律：

1. 对于每个实体，我们需要完成应用的基本功能，如 create, retrieve, update, and delete (CRUD).
2.我们需要每个实体持久化到数据库.
3.我们需要数据库为每个实体创建数据表。
4.我们需要安排实体之间的关系。

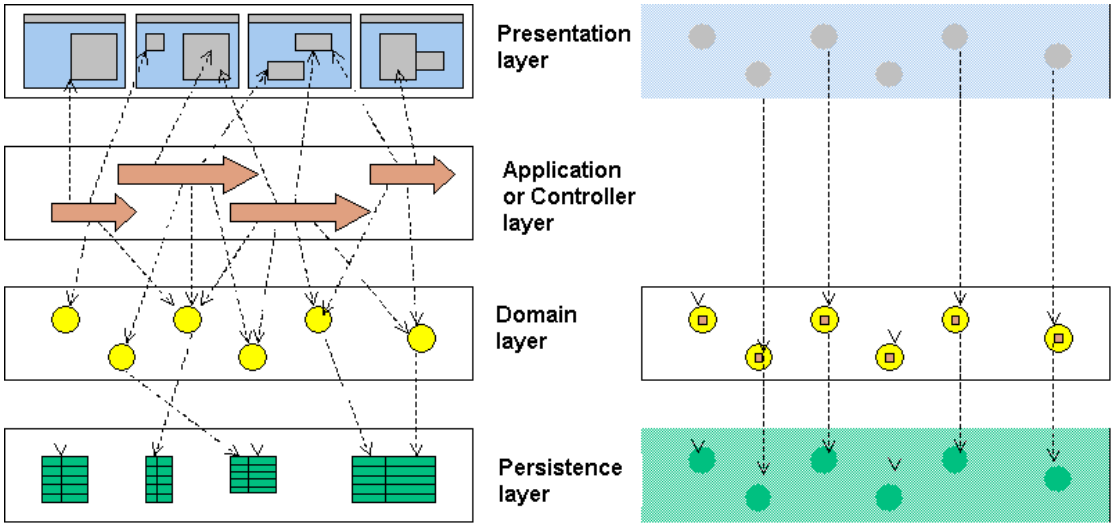
当然，在复杂应用中，不会只是这些功能，但是如果我们将这些功能通过框架实现，将大大提高我们的开发效率。

域驱动开发

域驱动开发(domain driven development framework)，简称 DDD 是一种最新的 OO 设计概念，它是由 ROR 和 Naked Object 组织提出的。

所谓 naked Object 是指一个复杂的域对象，这个 Object 是一个 POJO，但是不是一个傻傻的完全是属性的 POJO，而是封装了业务逻辑的 POJO，注意这里是最大的区别，一般业务逻辑我们是通过另外的 Service 类来实现，然后在 Service 中封装的 transaction script, 见 Martin Fowler 的 PoEEA. ，而 naked Object 则是合并起来的（有的类似回归传统了），个人感觉整个 OO 软件好像一直在玩 0 或 1 的游戏，不过也许最复杂的体系就是来自最简单的抉择，如股票/汇市等投资领域也如此。

naked Object 提出现在 J2EE 开发和裸体对象 DDD 开发下的图：



通过这张图我们可以看到，以前方式造成 J2EE 开发层次之间调用混乱，修改和拓展非常不方便，而在右边的 DDD 开发方式下，**界面(边界)对象就是域对象就是持久化的实体**，没有多余的 Contorller 或 Action 了。

现在怎么办?

ROR 提倡的 DDD 方式引起了众多 J2EE 开发者的兴趣，在各大 Java 媒体正在引起广泛的讨论，但是 ROR 不是 Java 的，那么有无基于 Java 的 DDD 开发框架呢？

目前有不少 DDD 开发框架正在诞生中，Jdon 框架正是在 ROR 这种精神指引下的一款快速开源开发框架，Jdon 框架 1.2.2 版本虽然不是一个完全意义上的 Naked Object，但是已经初步具备上图右边开发流程，具体可参考 Jdon Sample 的开发流程

使用 Jdon 框架开发 J2EE 应用系统，最重要的一个前提是：设计好你的域对象，然后在将域对象复制到表现层，变成表现层的 ActionForm/ModelForm；将域对象直接在持久层使用 Hibernate/iBatis 等持久化到数据库；如果使用 EJB 的实体 Bean 持久化技术，将无需实现建立数据表；应用系统部署时，J2EE 容器将直接根据实体建立数据表，也可节省前面步骤中两个步骤。

当然，目前 Jdon 框架是采取分离手法，遵循桥模式，将抽象和行为分离，每个域对象对应一个操作它的服务类或 DAO 类，服务类主要用来封装业务逻辑层，然后将业务 Service 作为一个业务组件暴露给表现层的 Controller/Action 类，而 Controller/Action 则无需代码，只要通过如下配置即可完成：

```
<model key="username" class="com.jdon.framework.samples.jpetstore.domain.Account">
    <actionForm name="accountForm"/> //指定边界类
    <handler>
      <service ref="accountService"> //指定某个业务接口
        <getMethod name="getAccount"/>
        <createMethod name="insertAccount"/> //业务接口的新增方法
        <updateMethod name="updateAccount"/> //业务接口的修改方法
```

```
            <deleteMethod name="deleteAccount"/>
        </service>
    </handler>
</model>
```

通过上述配置，净化了上图中应用控制层(Application or Controller layer)和 Domain Layer 之间对应关系，变得有条理而且明晰。

随着 Naked Object 被越来越多人认识和应用成熟，Jdon 框架也将转向支持 Naked Object。

Ioc 框架

代码案例

假设有调用者 B 和被调用者 A 代码如下：

```
调用者 B 类
 package test;
 public class B{
        AInfterface a;
        public B(AInfterface a){
            this.a = a
        }
        public void invoke(){
            a.myMethod();
        }
 }
被调用者 A 类：
 package test;
 public class A   implements AInfterface {
        public void myMethod(){
            System.out.println("hello");
        }
 }
```

生成 B 类实例代码如下：

```
B b = new B(new A());
```

创建 B 的实例要逐个照顾到 B 类中涉及到所有其他类（如 A 类）的实例化，给编程者带来代码编写的琐碎工作，无法提高效率。

使用 Jdon 框架的 Ioc 模式后，B 类生成实例代码如下：

```
B b =   (B)   WebAppUtil.getService("b");
b. invoke();
```

无需首先照顾其他类如 A 类的实例生成。B 的实例生成再也与其他类如 A 类没有任何关系了，实现松耦合。

实现上述调用效果，需另外实现 jdonframework.xml 配置如下：

```
<app>
    <services>
        <pojoService name="b"   class="test.B"/>
```

```
                <pojoService name="a"    class="test.A"/>
        </services>
        ……
    </app>
```

革命性优点

Java 编程中类创建成实例的过程简化：（Class -> Instance）

- 使用 Jdon 等 Ioc 框架前：

  编程者需要自己逐个解决这个 Class 相关涉及的其他 Class 的实例化。

- 使用 Jdon 等 Ioc 框架后：

  无需编程者自己实现这种级联式、琐碎的实例化过程。

  松耦合；更换各种子类方便。上例中，如果 Ainterface 有另外一个实现子类 AA 类，只要将 jdonframework.xml 中：

  <pojoService name="a"   class="test.A"/>

  更换为：

  <pojoService name="a"   class="test.AA"/>

Jdon 框架的特点？

Jdon 框架是一个真正轻量级别的开发框架，设计简单巧妙，适合快速开发各种架构的 J2EE 应用系统。它是一套符合当前国际水平的、面向构件开发的、国人拥有自主产权的中间件产品。

在 J2EE 应用开发中的主要优点

Jdon 框架给 J2EE 应用开发带来主要的好处是：

当你的项目刚开始时，它可能是一个小项目，实现一些简单功能，这时你可能只需要使用普通 JavaBeans（POJO）实现数据库操作业务，这个 POJO 一般一次请求生成一个实例（使用 new）。

当访问量逐渐增加，这种 POJO 每次请求生成和销毁都会耗费性能，你的 J2EE 应用系统可能出现性能降低缓慢等现象，这时我们就需要池（Pool）和缓存（Cache）来优化。

下面我们从 J2EE 应用系统运行原理开始简单分析：

如果在某个时刻有两个以上用户同时访问你的系统，也就是说同时发出请求（例如刷新页面），因为 J2EE 应用系统是运行在 J2EE 容器中（Tomcat JBoss），而 J2EE 容器（如 Tomcat）等前端有线程池支持；後端有数据库连接池支持，这些虽然提升了你的系统性能，但是因为你的代码最重的 POJO 是每次请求创建，这实际是整个系统的性能瓶颈。

用对象池优化你的 POJO 服务类；用缓存优化你的数据类。这就是使用 Jdon 框架带给你性能上的跳跃。

使用 Jdon 框架后，你的 J2EE 应用系统性能提升不少，但是访问量还是不断上升，尽管优化了其他该优化的：JVM；J2EE 服务器；数据库等，系统性能还是碰到了天花板。

使用 EJB 的多服务器集群分布式计算特性吧，只要增加服务器就可以提升性能。

这时，你可能用 EJB 将你的 POJO 封装起来，经过这样架构升级，如果你不使用 Jdon 框架，你的表现层（Struts）中原来调用 POJO 的代码需要修改，这是一个存在相当风险的大手术，你可能要全部重新测试；但是使用了 Jdon 框架，你做的只是在 Jdonframework.xml 配置中修改一下即可，整个系统代码无需更改。（其他框架升级到 EJB 时，需要你的 EJB 继承原来的接口，且需要 EJB 配置，不方便）

从你的项目一开始就使用 Jdon 框架，它带给了你方便的可伸缩的解决方案。也就是说：你的系统在规模很小时运行良好；在规模迅速扩张时，无需更改代码；带给你方便的架构更换。

Jdon 框架帮助你实现架构设计的可伸缩性。

相比其他框架，Jdon 框架实现了对 POJO Service 和 EJB 之间无缝支持。

Jdon 框架特点

Jdon 框架是根据最新设计思想 Ioc/AOP 构建的一个源框架（Meta Framework），随着时间推移，它将不断增加入新的设计概念和功能（如 MDA 或工作流引擎等）。

Jdon 框架给你的 J2EE 应用系统带来完整的高质量解决方案：

- Transparency（透明性）：框架配置修改维护方便，Jdon 框架配置划分三种：基础配

置组件、AOP 相关配置和应用服务配置，将经常需要修改与应用相关的配置单独出来，可分别修改和拓展。

- Scalability（可伸缩性）：使用本框架，可以开发出两种系统：真正轻量的 Web 应用系统或 Web+EJB 应用系统；无缝同时支持两种服务架构：EJB Session Bean 和 POJO Service（Web 应用），在不改变代码的情况下，可以很方便地将一个 Web 系统升级到 Web+EJB 系统。
- Performance（良好的性能）：Jdon 框架提供强大缓存功能，无需编程，在自己的系统中加入 Jdon 框架后，自动提升了每个应用系统的运行性能，特别是批量查询性能。对 Model 数据通过缓存拦截器提升性能;对 POJO 无态服务使用对象池拦截器;也实现了 POJO 的有态服务拦截器。
- High-Availability（高可用性）：在 J2EE 多层分离完全解耦的前提下，提供了数据增删改查（crud）快速开发方式，程序员需要编写的 crud 代码很少，表现层 crud 功能实现通常只需要配置就可以实现，No Code。如果持久层开放辅助以其他快速开发工具，可以迅速地提高 J2EE 开发速度。
- Extendable(扩展性)：Jdon 框架是可伸缩的、动态配置的，应用者可以将自己系统中的通用功能从具体系统中抽象出来，加入 Jdon 框架，从而逐步形成自己的行业专业开发框架。
- Loose coupling（松耦合）：使用 Jdon 框架可以完全解耦 J2EE 多层之间的耦合，从而实现应用系统稳定的健壮性、方便容易的可维护性。从而也使得 J2EE 表现层开发和业务层开发可以完全分离、各自独立同时进行，提高了开发效率。

Jdon 框架区别于其他同类框架的独特特点：

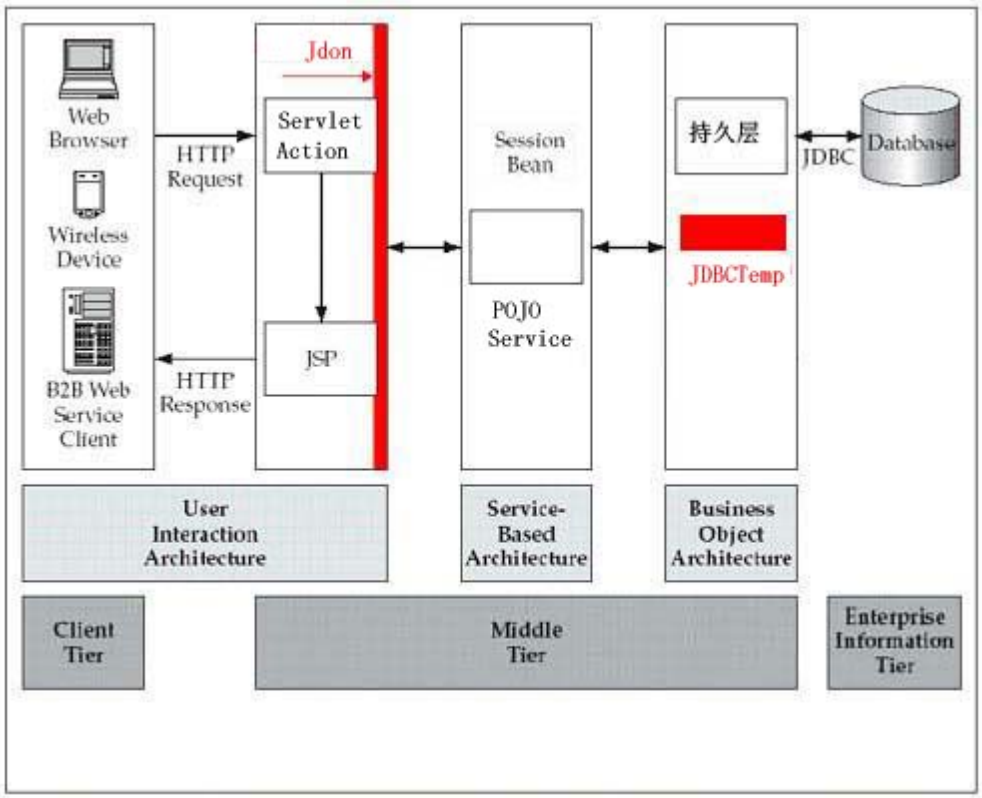
- Lightweight：真正轻量化，代码精简巧妙，是复杂庞大的 Spring 框架替代品。
- Auto Injection：自适应的构造器注射，Jdon 框架是基于 PicoContainer 为其微容器核心，PicoContainer 主要是构造器注射，而且它拥有强大的自适应注射；其它类似框架则要在配置文件中逐个明确指定注射对象，配置烦琐。
- EJB 服务平滑方便支持，Jdon 框架支持 EJB 服务就如同支持一般 POJO 服务一样，方便直接，无需 EJB 服务继承特定接口；而其它类似框架（如 Spring）则需要所有 EJB 服务继承特定接口。
- Jdon 框架的 AOP 功能是可分解的，通过缓存优化了动态代理实现，提高了运行性能，Jdon 框架 AOP 支持所有遵循继承 Aopalliance（如 Spring）的拦截器。
- Jdon 框架目前使用流行的 Struts 作为其主要表现层框架支持。

Jdon 框架与 J2EE 架构

目前在 J2EE 架构设计中主要分两大流派：EJB 和 POJO（普通的 Javabeans）；以 EJB 为代表的流派主流架构是：Struts/JSF+EJB 或者 Struts/JSF+Session Bean+hibernate/JDO；以 POJO 为代表的轻量流派则是：Struts/JSF+Hibernate/JDO。

Jdon 框架对这两种流派都有良好方便的支持，也可以在同一个系统混合这两种架构，所以可以形成：Struts/JSF+Jdon_EJB 或者 Struts/JSF+Jdon+Hibernate/JDO 等架构。

随着新技术诞生，目前它们组件形式无非是 EJB 或 POJO 两种，因此 Jdon 框架可适应这些未来的新技术实现。



适合哪些人

Jdon 框架主要是面向 J2EE 程序员，对于程序员要求并不很高，只要具备以下技术背景之一就可以尝试学习使用 Jdon 框架：

拥有 Jsp/Servlet JavaBeans J2EE 的 Web 编程经验的程序员

拥有 EJB 简单编程的程序员

最好有一些 Struts 感性认识和少量编程经验。

Jdon 框架安装说明

在 Jdon 框架源码包中的 dist 目录下，有下列几个包：

jdonFramework.jar	Jdon 框架核心包	必须需要
aopalliance.jar	AOPAlliance 包	必须需要
jdom.jar	读取 XML 的 JDOM 包	必须需要
picocontainer-1.1.jar	Picocontainer 包	必须需要
commons-pool-1.2.jar	Apache 对象池包	必须需要
log4j.jar	Log4j 调试记录跟踪包	可选
log4j.properties	Log4j 配置文件	可选
struts.jar …	struts 驱动包，支持 struts1.2	可选

Jdon 框架在 JBoss 中安装

安装步骤：

1. 确保已经安装 J2SE 1.4 以上版本，然后设置操作系统的环境变量 JAVA_HOME=你的 J2SE 目录

2. 下载 JBoss 3.X/JBoss 4.x

3. 安装 Jdon 框架驱动包：将 Jdon 框架源码包中的 dist 目录下除 log4j.jar 和 log4j.properties 以外的包拷贝到 jboss/server/default/lib 目录下。

4. 安装 struts 驱动包，下载 struts 1.2，将 jar 包拷贝到 jboss/server/default/lib。

或者使用 Jdon 框架例程 samples 中 SimpleJdonFrameworkTest 项目的 lib 目录。将该目录下 jar 包拷贝到 jboss/server/default/lib

对于具体 J2EE 应用系统，需要配置 Jboss 的数据库连接池 JNDI：

1.配置 JBoss 的数据库连接：

将数据库驱动包如 MYSQL 的 mysql-connector-java-3.0.14-production-bin.jar 或 Oracle 的 class12.jar 拷贝到 jboss/server/default/lib。

2. 选择 JBoss 的数据库 JNDI 定义文件：

在 jboss 的 docs 目录下寻找你的数据库的配置文件，如果是 MySQL，则是 mysql-ds.xml；如果是 Oracle；则是 oracle-ds.xml。下面以 MySQL 为例子。

将 mysql-ds.xml 拷贝到 jboss/server/default/deploy 目录下。

3. 修改配置数据库定义文件：

打开 jboss/server/default/deploy/mysql-ds.xml，如下：

```
<datasources>
  <local-tx-datasource>
    <jndi-name>DefaultDS</jndi-name> <!--  应用程序中使用 java:/DefaultDS 调用  -->
    <connection-url>jdbc:mysql://localhost:3306/test?useUnicode=true&amp;characterEncoding=UTF-8
```

```
                    </connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>root</user-name><!--  MySQL 数据库访问用户和密码，缺省是 root  -->
    <password></password>
  </local-tx-datasource>
  </datasources>
```

4.启动 JBoss

打开 jboss/server/default/log/server.log 如果没有错误，一切 OK，一般可能是数据库连接错误，检查 mysql-ds.xml 配置，查取相关资料，弄懂每行意义。

至此，可以将基于 Jdon 框架开发的 J2EE 应用程序部署到 JBoss 中。

一般是将*.ear 或*.war 拷贝到 jboss/server/default/deploy 目录下即可。

Jdon 框架在其他商业服务器中安装

只要将 Jdon 框架包和 struts 1.2 包安装到服务器的库目录下即可，或者配置在系统的 classpath 中即可。如果你的服务器没有 log4j 包，那么还需要 log4j.jar，并将 log4j.properties 放置在系统 classpath 中。

Jdon 框架在 Tomcat 中安装

Jdon 框架在 Tomcat 下安装主要问题是 log4j 问题，下面是安装步骤：

1. 将 struts 驱动包和 Jdon 框架包（包括 log4j.jar）拷贝到 tomcat/ common/lib 目录下。
2. 将 Jdon 框架源码包 dist 目录下的 log4j. properties 拷贝到 tomcat/common/classes 目录。
3. 配置 Tomcat 中运行 log4j 的关键是：检查 commons-logging.jar 和 log4j.jar 文件在 common/lib 目录，struts 驱动包中已经包含 commons-logging.jar 包。
4. 上面步骤都正常了，可以启动 Tomcat，但是你会发现 tomcat/logs 下没有输出记录，因为我们已经使用新的 log4j，所以为了使得 tomcat 运行信息输出文件便于调试，编辑/common/classes 下 log4j. properties：
   将#log4j.appender.R.File=D:/javaserver/jakarta-tomcat-5.0.28/logs/tomcat.log 一行前面的#删除，文件目录是绝对路径，更改为你自己的目录和文件。
   同时将 log4j.rootLogger=INFO, A1 一行前面加#注释。
   重新启动 Tomcat，这时可以从 tomcat.log 看到输出记录。
5. 配置 Jdon 框架运行过程输出，在 log4j. properties 中下面一行：
   log4j.logger.com.jdon=DEBUG
   该配置将会显示 Jdon 框架的主要运行信息，如果你要关闭，只要更改如下：
   log4j.logger.com.jdon=ERROR

基本概念和配置篇

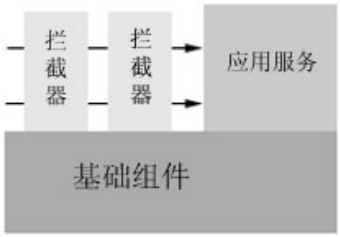
本章节讲述 Jdon 框架的基本配置和一些基本设计，是使用 Jdon 框架之前需要了解的章节，通过这些基本概念说明，可以让程序员对 Jdon 框架有一个大体概念上的轮廓。

本章节对于初学者不是必读，其中很多用法可在后面案例中学习，因此，本章节也可以直接跳过去，进入下一个章节阅读。

可彻底分离的组件管理

Jdon 框架可以实现几乎所有组件可配置、可分离的管理，这主要得益于 Ioc 模式的实现，Jdon 框可以说是一个组件（JavaBeans）管理的微容器。

在 Jdon 框架中，有三种性质的组件（JavaBeans）：框架基础组件；AOP 拦截器组件和应用服务组件。三种性质的组件都是通过配置文件实现可配置、可管理的，框架应用者替换这三种性质组件的任何一个。



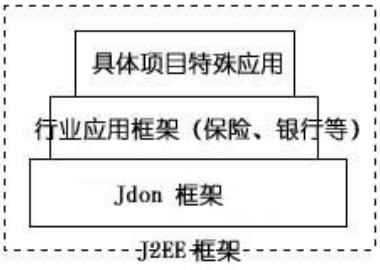
框架基础组件是 Jdon 框架最基本的组件，是实现框架基本功能的组件，如果框架应用者对 Jdon 框架提供的功能不满意或有意替换，可以编写自己的基础功能组件替代，从而实现框架的可彻底分离或管理。

应用服务组件是框架应用者针对具体项目设计的组件，如一个新闻系统，用户管理 UserDao、新闻管理 NewsManager 都属于应用服务组件。

AOP 拦截器组件主要是指一些应用相关的通用基础功能组件，如缓存组件、对象池组件等。相当于应用服务组件前的过滤器（Filter），在客户端访问应用服务组件之前，必须首先访问的组件功能。

这三种性质组件基本函括了应用系统开发大部分组件，应用服务组件是应用系统相关组件，基本和数据库实现相关，明显特征是一个 DAO 类；当应用服务组件比较复杂时，我们就可以从中重整 Refactoring 出一些通用功能，这些功能可以上升为框架基础组件，也可以抽象为 AOP 拦截器组件，主要取决于它们运行时和应用服务组件的关系。当然这三种性质框架组件之间可以相互引用（以构造方法参数形式），因为它们注册在同一个微容器中。

使用 Jdon 框架，为应用系统开发者提炼行业框架提供了方便，框架应用者可以在 Jdon 框架基本功能基础上，添加很多自己行业特征的组件，从而实现了框架再生产，提高应用系统的开发效率。



Jdon 框架应用第一步

在 JdonFramework 中，所有组件都是在配置文件中配置的，框架的组件是在 container.xml 和 aspect.xml 中配置，应用系统组件是在 jdonframework.xml 中配置，应用系统组件和框架内部或外部相关组件都是在应用系统启动时自动装载入 J2EE 应用服务器中，它们可以相互引用（以构造器参数引用，只要自己编写的普通 JavaBeans 属于构造器注射类型的类就可以），好似是配置在一个配置文件中一样。

因此，组件配置主要有三个配置文件：应用服务组件配置 container.xml、AOP 拦截器组件 aspect.xml 和应用服务组件配置 jdonframework.xml

初次使用 Jdon 框架时，需要在应用系统中指定自己定义的配置文件，分两步：

1. 定义自己的 jdonframework.xml 配置文件，这是必须的步骤。

目前 Jdon 框架中整合了 Struts 前台表现层技术，因此可以通过 Struts 的 Plugin 实现 jdonframework.xml 启动，这样做的好处，可以实现 Struts 多模块开发，一个 Struts 项目中可能有多个功能模块，每个功能模块涉及从页面表现(struts 配置和 tiles 配置)、模型设计以及后台持久化等横向一系列组件，Jdon 框架通过 Plugin 启动 jdonframework.xml 可以支持这种多模块开发方式。

Jdon 框架已经提供一个 Plugin 缺省实现子类：com.jdon.strutsutil. InitPlugIn，你可以根据自己要求实现自己的 Plugin 实现子类。在 struts-config.xml（或其它 struts 模块配置文件如 struts-config-admin.xml 等等）中配置 Plugin 实现子类：

```
<plug-in className="com.jdon.strutsutil.InitPlugIn">
    <set-property property="modelmapping-config" value="news.jdonframework.xml" />
</plug-in>
```

InitPlugIn 主要实现从 struts-config.xml 中读取 modelmapping-config 的值，modelmapping-config 的值是你自己定义的 jdonframework 应用配置文件，文件名不一定是

jdonframework.xml，可以任意指定，例如 news.jdonframework.xml 表示 jdonframework.xml 在包名 news 下，com.jdon.app.myframework.xml 表示 myframework.xml 在包名 com.jdon.app 下。

2. Jdon 框架启动时将使用 jdonframework.jar 中 META-INF 目录下缺省的 container.xml 和 aspect.xml。如果你需要拓展 Jdon 框架，如自己开发了一些小零碎组件，如计算公式、报表组件等或者行业软件的一些通用功能，这些组件可能需要启动时就载入，或者希望它们实现可配置可替换，那么就将它们整合入 Jdon 框架中，有两种加入自己配置的方式（该步骤不是必需的）：

第一. 定义自己的组件配置文件和拦截器组件配置，文件名必须为 mycontainer.xml 和 myaspect.xml，这两个文件必须放置在系统的 classpath 路径中，或者必须在你自己的 jar 包，这个 jar 包可以和 Jdon 框架 jar 包一起部署。

第二. 可以在应用系统的 web.xml 中定义，可以在 web.xml 中加入自己定义这两种配置：

```
<context-param>
    <param-name>containerConfigure</param-name>
    <param-value>/WEB-INF/mycontainer.xml</param-value>
    <param-name>aspectConfigure</param-name>
    <param-value>/WEB-INF/myaspect.xml</param-value>
</context-param>
<listener>
     <listener-class>com.jdon.container.startup.ServletContainerListener</listener-class>
</listener>
```

那么你的 mycontainer.xml 和 myaspect.xml 必须放置在 Web 项目的 WEB-INF 目录下，当然这两个文件名可以自己任意取名。

用户自己定义的配置文件中可以覆盖缺省的 container.xml 或 aspect.xml 相应的配置。只要取相同的 name 值就可以。

应用服务组件配置

jdonframework.xml 是应用服务组件配置文件，文件名可自己自由定义，jdonframework.xml 中主要是定义 Model（模型）和 Service（服务）两大要素。

jdonframework.xml最新定义由http://www.jdon.com/jdonframework.dtd规定。

<models>段落是定义应用系统的建模，一个应用系统有哪些详细具体的模型，可由 Domain Model 分析设计而来。<models>中的详细配置说明可见 数据模型增、删、改、查章节。

<services>段落是定义服务组件的配置，目前有两种主要服务组件配置：EJB 和 POJO。

EJB 服务组件配置如下：

```
<ejbService name="newsManager">
        <jndi name="NewsManager" />
        <ejbLocalObject class="news.ejb.NewsManagerLocal"/>
</ejbService>
```

每个 ejbService 组件有一个全局唯一的名字，如 newsManager，有两个必须子定义：该 EJB 的 JNDI 名称和其 Local 或 remote 接口类。

POJO 服务组件配置如下：

```
<pojoService name="userJdbcDao" class="news.container.UserJdbcDao">
        <constructor value="java:/NewsDS"/>
```

```
</pojoService>
```

POJO 服务也必须有一个全局唯一名称，如 userJdbcDao，以及它的 class 类定义。如果该 POJO 构造器有字符串变量，可在这里定义其变量的值，目前 Jdon 框架只支持构造器字符串变量注射。

如果该 POJO 服务需要引用其它服务，例如 UserPrincipalImp 类的构造器如下：

```
public UserPrincipalImp(UserDao userDao){

    this.userDao = userDao;

}  ……
```

UserPrincipalImp 构造器需要引用 UserDao 子类实现，只需在 jdonframework.xml 中同时配置这两个服务组件即可，Jdon 框架会自动配置它们之间的关系：

```
<pojoService name="userJdbcDao" class="news.container.UserJdbcDao">
        <constructor value="java:/NewsDS"/>
</pojoService>
<pojoService name="userPrincipal" class="news.container.UserPrincipalImp"/>
```

上面配置中 news.container.UserJdbcDao 是接口 UserDao 的子类实现，这样，直接通过 userPrincipal 这个名称可获得 UserPrincipalImp 的实例。

基础组件配置说明

container.xml 是 Jdon 框架基础组件配置文件，container.xml 中包含的组件将由 Jdon 框架在启动时向微容器（PicoContainer）中注册，至于这些组件之间的依赖关系由微容器解决，称为 Ioc 模式。

container.xml 内容主要由每行配置组成，每行格式如下：

```
<component name="组件名称" class="POJO 类名称" />
```

如

```
<component name="modelHandler" class="com.jdon.model.handler.XmlModelHandler" />
```

代表组件 com.jdon.model.handler.XmlModelHandler，其名称为 modelHandler，如果需要在程序中调用 XmlModelHandle 实例，只需要以 modelHandler 为名称从微容器中获取即可。

组件配置也可以带有参数，例如下行：

```
    <component name="cache" class="com.jdon.controller.cache.LRUCache" >
        <constructor value="cache.xml"/>
    </component>
```

而 LRUCache 的类代码如下：

```
public class LRUCache implements Cache {
        public LRUCache(String configFileName) {
            PropsUtil propsUtil = new PropsUtil(configFileName);
            cache = new UtilCache(propsUtil);
        }
      ……..
}
```

这样 LRUCache 中的 configFileName 值就是 cache.xml，在 cache.xml 中定义了有关缓存的一些设置参数。目前 Jdon 框架只支持构造器是纯字符串型，可多个字符串变量，但不能字类型和其它类型混淆在一起作为一个构造器的构造参数。如果需要多个类型作为构造参数，可新建一个包含字符串配置的类，这个类就可和其它类型一起作为一个构造器的构造参

数了。

一般在 container.xml 中的组件是框架基本功能的类，不涉及到具体应用系统。

拦截器组件配置说明

aspect.xml 是关于拦截器组件配置，有两个方面：advice（拦截器 Interceptor）和 pointcut（切入点）两个方面配置，有关 AOP 的基本概念可见：http://www.jdon.com/AOPdesign/aspectJ.htm。

Jdon AOP 的设计目前功能比较简单，不包括标准 AOP 中的 Mixin 和 Introduction 等功能；Pointcut 不是针对每个 class 和方法，而是针对一一系列 class，拦截粒度最粗，与许多复杂完整的 AOP 框架（如 AspectJ、Spring）不同的是：Jdon AOP 在粒度方面是最粗的，AspectJ 最细，Spring 中等，如果你需要粒度细腻的 AOP 功能，还是推荐使用 Spring 或 AspectJ。目前这样设计是主要有两个原因：

每个类在运行时刻都实现动态拦截，在性能上有所损失，这如同职责链模式缺点一样。

在实际应用中，可以通过代理模式 Proxy、装饰模式 Decorator 实现一些细腻度拦截，结合容器的 Ioc 特性，这两个代理模式使用起来非常方便，运行性能有一定提高。

Jdon AOP 主要针对拦截器 Interceptor 设计，它可以为所有 jdonframework.xml 中定义的 Service 提供拦截器；所有的拦截器被放在一个拦截器链 InterceptorsChain 中。

Jdon AOP 并没有为每个目标实例都提供拦截器配置的功能，在 JdonAOP 中，目标对象是以组为单位，而非每个实例，类似 Cache/Pool 等这些通用拦截器都是服务于所有目标对象。

JdonAOP 拦截器目标对象组有三种：全部目标服务；EJB 服务；POJO 服务（EJB 服务和 POJO 服务是在 JdonFramework.xml 中定义的 ejbService 和 pojoService）。从而也决定了 Pointcut 非常简单。以下是 aspect.xml 中的配置：

```
<interceptor name="cacheInterceptor"
                class="com.jdon.aop.interceptor.CacheInterceptor"
                pointcut="services" />
```

其中 pointcut 有三种配置可选：services ; pojoServices 和 ejbServices

拦截器配置也可以如组件配置一样，带有 constructor 参数，以便指定有关拦截器设置的配置文件名。

拦截器的加入不只是通过配置自己的 aspect.xml 可以加入，也可以通过程序实现，调用 WebAppUtil 的 addInterceptor 方法即可，该方法只要执行一次即可。

如何实现自己的拦截器？

以对象池拦截器 PoolInterceptor 为例，对象池是使用 Aapche Commons Pool 开源产品，对象池主要是为了提高 POJO Service 的运行性能，在没有对象池的情形下，POJO Service 每次被访问时，要产生一个新的实例，如果并发访问用户量很大，JVM 将会频繁创建和销毁大量对象实例，这无疑是耗费性能的。

使用对象池则可以重复使用一个先前以前生成的 POJO Service 实例，这也是 Flyweight 模式一个应用。

对象池如何加入到 Jdon 框架中呢？有两种方式：1.替代原来的 POJO Service 实例创建方式，属于 PojoServiceFactory 实现（Spring 中 TargetSource 实现）；2.通过拦截器，拦截在原来的 PojoServiceFactory 实现之前发生作用，同时屏蔽了原来的 PojoServiceFactory 实现。

Jdon 框架采取的这一种方式。

首先，为拦截器准备好基础组件。对象池拦截器有两个：对象池 com.jdon.controller.pool.CommonsPoolAdapter 和对象池工厂 com.jdon.controller.pool.CommonsPoolFactory，这两个实现是安装 Apache Pool 要求实现的。

第二步，需要确定拦截器的 Pointcut 范围。前面已经说明，在 Jdon 框架中有三个 Pointcut 范围：所有服务、所有 EJB 服务和所有 POJO 服务，这种划分目标的粒度很粗糙，我们有时希望为一些服务群指定一个统一的拦截器，例如，我们不想为所有 POJO 服务提供对象池，想为指定的一些目标服务（如访问量大，且没有状态需要保存的）提供对象池，那么如何实现呢？这实际是如何自由划分我们自己的目标群（或单个目标实例）的问题

我们只要制作一个空接口 Poolable，其中无任何方法，只要将我们需要对象池的目标类实现这个接口即可，例如 com.jdon.security.web.AbstractUserPrincipal 多继承一个接口 Poolable，那么 AbstractUserPrincipal 所有的子类都将被赋予对象池功能，所有子类实例获得是从对象池中借入，然后自动返回。

这种通过编程而不是配置实现的 Pointcut 可灵活实现单个目标实例拦截或一组目标实例拦截，可由程序员自由指定划分，非常灵活，节省了琐碎的配置，至于 Pointcut 详细到类方法适配，可在拦截器中代码指定，如缓存 com.jdon.aop.interceptor.CacheInterceptor 只拦截目标服务类中的 getXXXX 方法，并且该方法的返回结果类型属于 Model 子类，为了提高性能，CacheInterceptor 将符合条件的所有方法在第一次检查合格后缓存起来，这样，下次无需再次检查，省却每次检查。

第三步，确定拦截器在整个拦截器链条中的位置。这要根据不同拦截器功能决定，对象池拦截器由于是决定目标服务实例产生方式，因此，它应该最后终点，也就是在拦截器链中最后一个执行，aspect.xml 中配置拦截器是有先后的：

```
<interceptor name="cacheInterceptor"
                class="com.jdon.aop.interceptor.CacheInterceptor" pointcut="services" />

<interceptor name="poolInterceptor"
                class="com.jdon.aop.interceptor.PoolInterceptor" pointcut="pojoServices" />
```

拦截器链中排序是根据 Interceptor 的 name 值排序的，cacheInterceptor 第一个字母是 c，而 poolInterceptor 第一个字母是 p，按照字母排列顺序，cacheIntercepotr 排在 poolInterceptor 之前，在运行中 cacheInterceptor 首先运行，在以后增加新拦截器时，要注意将 poolInterceptor 排在最后一个，name 值是可以任意指定的，如为了使 PoolInterceptor 排在最后一个，可命名为 zpoolInterceptor，前面带一个 z 字母。

第四步，确定拦截器激活行为是在拦截点之前还是之后，或者前后兼顾，这就是 advice 的三种性质：Before、After 或 Around，这分别是针对具体拦截点 jointcut 位置而言。

虽然 Jdon 框架没有象 Spring 那样提供具体的 BeforeAdvice 和 AfterReturningAdvice 等接口，其实这些都可以有程序员自己直接实现。

Jdon 框架的拦截器和 Spring 等遵循 aopalliance 的 AOP 框架继承同一个接口 MethodInterceptor，也就是说，一个拦截器在这些不同 AOP 框架之间可以通用，具体实现方式不同而已。

例如，一个 AroundAdvice 实现如下代码，它们都只需要完成 invoke 方法内尔：

```
public class AroundAdvice      implements MethodInterceptor
{
    public Object invoke( MethodInvocation invocation)    throws Throwable
    {
        System.out.println("Hello world! (by " + this.getClass().getName() + ")");
```

```
            invocation.proceed();
            System.out.println("Goodbye! (by " + this.getClass().getName() + ")");
            return null;
        }
    }
```

beforeAdvice 实现代码如下：

```
public class BeforeAdvice      implements MethodInterceptor
{
    public Object invoke( MethodInvocation invocation)    throws Throwable
    {
        System.out.println("Hello world! (by " + this.getClass().getName() + ")");
        invocation.proceed();
    }
}
```

由此可以注意到，invocation.proceed()类似一个joincut点，这个方法类似 400 米接力比赛中的传接力棒，将接力棒传到下一个拦截器中，非常类似与servletFilter中的chain.doFilter(request, response);拦截器一些更详细说明可参考Spring相关文档，如下面网址：http://www.onjava.com/pub/a/onjava/2004/10/20/springaop2.html，和Jdon拦截器原理基本一致。

考察对象池拦截器功能，它实际是一个 around advice，在 joincut 之前需要从对象池借用一个目标服务实例，然后需要返回对象池。com.jdon.aop.interceptor.PoolInterceptor 主要核心代码如下：

```
Pool pool = commonsPoolFactory.getPool(); //获得对象池
Object poa = null;
Object result = null;
 try {
      poa = pool.acquirePoolable(); //借用一个服务对象
      Debug.logVerbose(" borrow a object:" + targetMetaDef.getClassName()
                              + " from pool", module);
      //set the object that borrowed from pool to MethodInvocation
      //so later other Interceptors or MethodInvocation can use it!
      proxyMethodInvocation.setThis(poa);   //放入 invocation，以便供使用
      result = invocation.proceed();
} catch (Exception ex) {
      Debug.logError(ex, module);
 } finally {
      if (poa != null) {
         pool.releasePoolable(poa);   //将服务对象归还对象池
         Debug.logVerbose(" realease a object:" + targetMetaDef.getClassName()
                                    + " to pool", module);
      }
 }
 return result;
```

经过上述四步考虑，我们基本可以在 Jdon 框架动态 plugin 自己的拦截器，关于对象池拦截器有一点需要说明的，首先 EJB 服务因为有 EJB 容器提供对象池功能，因此不需要对象池了，在 POJO 服务中，如果你的 POJO 服务设计成有状态的，或者你想让其成为单例，就不能使用对象池，只要你这个 POJO 服务类不继承 Poolable，它的获得是通过组件实例方式获得，参考后面“组件实例和服务实例”章节。

POJO 应用篇

本章节主要讲述如何在应用系统中应用 Jdon 框架，技术架构主要是基于 J2EE 的 Web 结构，使用普通 JavaBeans 实现服务功能情况下的 Jdon 框架使用，POJO 应用架构适合 Struts+Jdon+Hibernate 等架构。

组件实例和服务实例

在 Jdon 框架中， POJO 实例分为两种性质：组件实例和服务实例。

组件实例是指那些注册在容器中的普通 Javabeans，它们是单例的，也就是你每次获得的组件实例都是同一个实例，也就是说是单态的。

组件 POJO 获得方法是通过 WebAppUtil 的 getComponentInstance 方法，例如在 container.xml 中有如下组件定义：

```
<component name="modelManager" class="com.jdon.model.ModelManagerImp" />
```

在程序中如果要获得 ModelManagerImp 组件实例的方法是：

```
ModelManager modelManager =
                (ModelManager)WebAppUtil.getComponentInstance(“modelManager”, sc);
```

组件实例获得的原理实际是直接在微容器中寻找以前注册过的那些 POJO，相当于直接从 XML 配置中直接读取组件实例配置。

服务实例是指在 jdonframework.xml 中定义的 ejbService 和 pojoService，当然它们也可以组件实例方式获得，但是如果以组件实例方式获得，AOP 功能将失效；而以服务实例方式获得的话，在 aspect.xml 中定义的拦截器功能将激活。

EJB 服务一定是通过服务实例方式获得，只有普通 JavaBeans（POJO）才可能有这两种方式。

因此，除非特殊需要，一般推荐在应用系统中，通过获得服务实例方式来获得 jdonframework.xml 中定义的服务实例。

如在 jdonframework.xml 中有如下服务组件定：

```
<pojoService name="userJdbcDao" class="news.container.UserJdbcDao">
        <constructor value="java:/NewsDS"/>
</pojoService>
```

获得服务实例的方法代码如下：

```
UserDao ud = (UserDao)WebAppUtil.getService(“userJdbcDao”, request);
```

UserDao 是 UserJdbcDao 的接口。注意，这里必须 getService 结果必须下塑为接口类型，不能是抽象类或普通类，这也是与 getComponentInstance 不同所在。

如果你在 aspect.xml 将 pojoServices 都配置以对象池 Pool 拦截器，那么上面代码将是从对象池中获取一个已经事先生成的实例。

如何获得一个 POJO 实例？

在上面章节说明了服务实例和组件实例的区别，在 jdonframework.xml 中配置 POJO 既可以组件实例获得，也可以服务实例获得。

首先，我们需要在 jdonframework.xml 中定义自己的 POJO 服务，例如在

jdonframework.xml 有如下一行定义：

```
<pojoService name="userJdbcDao" class="news.container.UserJdbcDao">
```

那么在应用程序中需要访问 UserJdbcDao 实例有以下两种方式：

第一. 通过 WebAppUtil 工具类的 getService 方法获得服务实例，如：

```
UserDao ud = (UserDao)WebAppUtil.getService("userJdbcDao", request);
```

getService 方法每次返回的一个新的服务实例对象，相当于 new 一对象。如果对象池拦截器被配置，那么这里返回的就是从对象池中借用的一个对象。

第二. 通过 WebAppUtil 工具类的 getComponentInstance 方法获得组件实例，这也是获得 UserJdbcDao 一个实例，与服务实例不同的是，每次获得组件实例是同一个对象，因此，如果这个服务中如果包含上次访问的状态或数据，下次访问必须使用到这些状态和数据，那么就必须使用 getComponentInstance 方法获得服务实例。

注意，以上方式是假定你获得一个 POJO 实例，是为了使用它，也就是说，是为了访问它的方法，如访问 userJdbcDao 的 getName 方法，就要使用上述方式。

如果你不是为了使用它，而是作为别的 POJO 服务的输入参数，如构造器的输入参数，那么完全不必通过上述方式，你只要直接使用上述方式获得那个 POJO 服务的实例就可以，因为容器自动完成它们的匹配。

还有一点要求注意的是：使用 getService 获得服务实例，必须该服务类有一个接口，这样才能将 getService downcasting 下塑为其接口，否则只能是一个普通 Object，你获得后无法使用它。当然 getComponentInstance 没有这样限制。

如何编写一个 POJO 类？

既然在 Jdon 框架中获得 POJO 服务这么方便，那么 POJO 服务类的编写是否有特殊规定，回答是没有，就是一个普通的 Java 类，当然如果你需要在这个类引用其他类，最好将其他类作为构造器参数，如 A 类中引用 B 类，A 类的写法如下：

```
class A {
      private B b;
      public A(B b){
            this.b = b;
      }
      ….
}
```

这样，在 jdonframework.xml 中配置如下两行：

```
<pojoService name=" a" class="A">
<pojoService name=" b" class="B">
```

当然，你也可以使用下面一种方式：

```
 class A {
      private B b    = new B();
      ….
}
```

这样就无需配置 jdonframework.xml，但是这样做的缺点是不够灵活，万一 B 类改名或更改后就需要修改 A 类代码，带来强烈的耦合性，这也是为什么使用 Ioc 模式的原因。

如果你希望你的 POJO 服务能够以对象池形式被访问，那么你的类需要 implements

com.jdon.controller.pool.Poolable

如何获得一个 POJO 服务的运行结果

在应用系统中，我们不但可以通过上面方式获得一个 POJO 实例，然后在通过代码调用其方法，获得其运行结果，例如写入代码：

userJdbcDao.getName();

可以获得 getName 方法的运行结果。

除此之外，Jdon 框架还可以直接获得 POJO 服务的运行结果，只要你告诉它 POJO 类名、需要调用的方法名和相关方法参数类型和值，借用 Java Method Relection 机制，Jdon 框架可以直接获得运行结果。

实现这个功能，只要和接口 com.jdon.controller.service.Service 打交道即可：

```
public interface Service {

    public Object execute(String name,
                          MethodMetaArgs methodMetaArgs,
                          HttpServletRequest request)    throws Exception;


    public Object execute(TargetMetaDef targetMetaDef,
                          MethodMetaArgs methodMetaArgs,
                          HttpServletRequest request)    throws Exception;


}
```

Service 提供了两种获得某个 POJO 服务运行结果的方法。一个是以 Jdonframework.xml 中配置的 POJO 服务名称为主要参数，这是经常使用的一个情况。

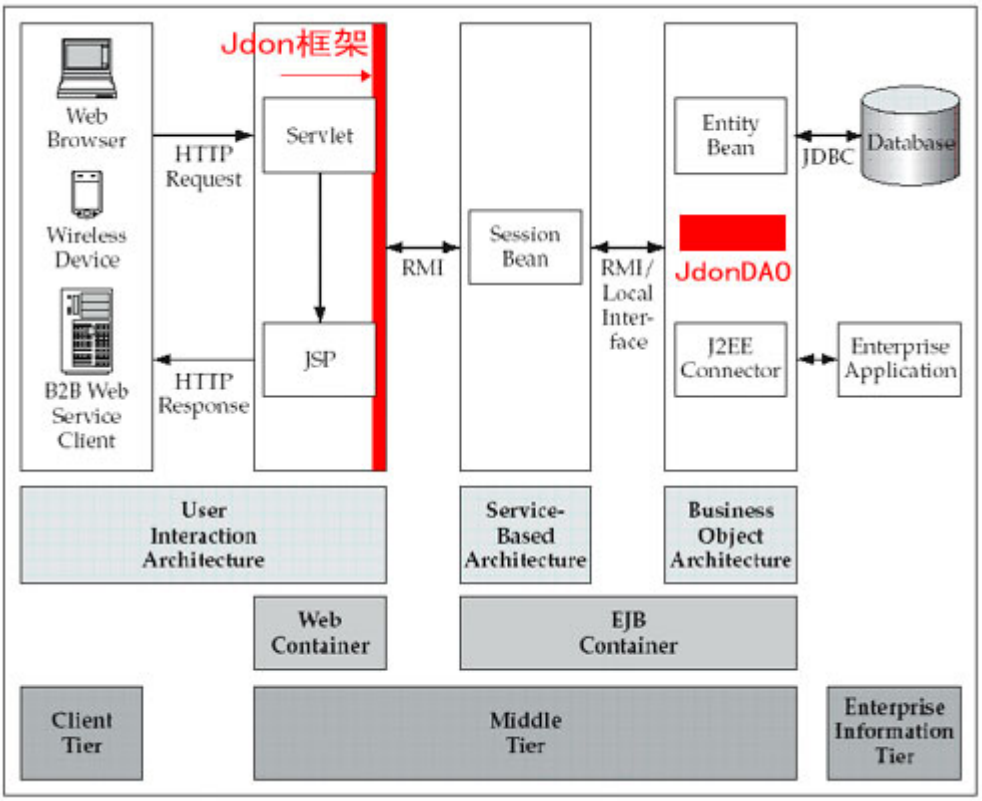
MethodMetaArgs 是包含调用的方法名、方法参数类型和方法参数值。

这种调用方式适合于 POJO 服务配置式调用，也就是说，通过编写自己的 XML 配置文件也可以实现如同写代码一样的服务调用和运行。

EJB 应用篇

Jdon 框架不只可以支持普通 JavaBeans，也就是 POJO 结构，也支持 EJB 架构，使用 EJB 的好处是能够获得可伸缩的、分布式的强大计算性能，当然 EJB 的开发需要借助 JBuilder 之类商业开发工具的图形开发功能才能方便快速实现。

使用 Jdon 框架可以开发出基于 Struts+Jdon+EJB 的标准 J2EE 架构系统，Jdon 框架在标准的 J2EE 架构中所处位置如下图的红色标记：



从上图底部向上看，是 J2EE 架构从抽象到具体技术的演变划分，J2EE 是一个中间件系统，那么中间层包括哪些部分呢？主要区分 Web 层和 EJB 层，Web 层主要是完成用户交互操作功能，Jdon 框架主要部分就位于这个 Web 层最後端，直接和 EJB 层打交道；同时 Jdon 框架有一部分 JdonDAO 运行在 EJB 容器中。

如何获得一个 EJB 服务实例？

如何在表现层如 Struts 中获得一个 EJB 实例呢？

如同获得 POJO 实例一样，也是通过 WebAppUtil 工具类的 getService 方法获得，如果在 jdonframework.xml 中有如下配置：

```
<ejbService name="newsManager">
        <jndi name="NewsManager" />
        <ejbLocalObject class="news.ejb.NewsManagerLocal"/>
</ejbService>
```

那么，通过下面代码就可以访问 NewsManagerLocal：

```
NewsManagerLocal nm = (NewsManagerLocal)WebAppUtil.getService(“newsManager”, request);
```

这样就访问 nm 这个 EJB 对象的方法了。

如何编写一个 EJB 类？

Jdon 框架对于一个 EJB 类的编写没有任何约束和规定。

但是，如果你原来使用 POJO 服务实现你的服务层，想无缝迁移到 EJB 服务，那么此时你的 Session Bean 需要继承 implements 原来 POJO 服务的接口，同时在 jdonframework.xml 的 ejbService 加入 interface 配置，指定原来 POJO 服务接口类，这样才能保证原来代码中通过 getService 方法获得服务实例的调用代码无需改变：

```
<ejbService name="testService2" >
        <jndi name="TestEJB" />
        <ejbLocalObject class="com.jdon.framework.test.ejb.TestEJBLocal"/>
        <interface class="com.jdon.framework.test.service.TestService" />
      </ejbService>
```

如何获得 EJB 服务运行结果？

使用方式和获得 POJO 服务一样。

这种方式运行原理简要如下：当知道一个 EJB/POJO 的接口，通过 Proxy.newProxyInstance 生成一个动态代理实例（InvocationHandler 的实现子类）即可，以后对 EJB/POJO 的调用，实际由这个动态代理实例的 invoke 自动激活，从而使用 Method Reflection 实现 EJB/POJO 的调用。

增删改查(crud)快速开发

Jdon 框架提供了基于 Struts 的快速开发，可以快速开发出数据的增删改查功能（crud），以及批量查询等功能。

本章是与 Jdon 框架其它功能分离的，你可以只使用 Jdon 框架的组件管理和 AOP 功能，而不必使用基于 Struts 的快速开发功能，随着新的表现层技术出现，Jdon 框架将推出基于的技术如 JSF 的快速开发框架。

关于为什么使用 jdon 框架的 crud 功能，除了在开发速度上有极大提高以外，在设计上也有很多优点：

边界类，控制类和业务接口的关系：

http://www.jdon.com/jive/article.jsp?forum=91&thread=21245

J2EE 中几种业务代理模式的实现和比较:

http://www.jdon.com/artichect/businessproxy.htm

struts 基础

Jdon 框架的 crud 功能是基于 struts 实现，主要工作是表现层的配置，那么是否需要程序员熟悉 struts 呢？其实完全不必。

但是，需要了解 struts 的一些基础：struts 主要有一个配置文件，在 Web 目录的 WEB-INF 下 struts-config.xml，其中主要有两部分配置：ActionForm 和 Action。

ActionForm 配置：主要是配置你设计的 ActionForm 子类；例如：

```
<struts-config>
  <form-beans>
    <form-bean name="accountForm"
                type="com.jdon.framework.samples.jpetstore.presentation.form.AccountForm"/>
  </form-beans>
  …..
</struts-config>
```

Action 配置：相当于一个 servlet，也需要在 struts-config.xm 中 action-mappings l 配置，例如：

```
<action-mappings>
    <action name="accountForm" path="/shop/newAccountForm"
                                  type="com.jdon.strutsutil.ModelViewAction" scope="request">
      <forward name="create" path="/account/NewAccountForm.jsp" />
    </action>
</action-mappings>
```

action 中配置简要说明：

name 是 ActionForm 名称，是前面 ActionForm 名称；

path 是该 action 在浏览器中调用的 url 名称，使用 http://xxxx/web 名称/shop/newAccountForm.do 就可以调用这个action，在path值后面加一个.do即可，.do是在web.xml中配置的：

```
  <servlet-mapping>
```

```
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
```

你可以改变*.do配置，改为*.shtml，那么上面action(Servlet)调用变成http://xxxx/web名称/shop/newAccountForm.shtml

type 是你继承 struts 的 Action 实现子类：如 com.jdon.strutsutil.ModelViewAction 是 Jdon 框架的实现子类。

Scope 是值你的 ActionForm 生存周期，一般有 request 或 session，相当于 Jsp 中 useBean 中的 scope，表示 ActionForm 实例有效范围，request 表示用户发出一个请求周期；session 表示是从用户登陆到退出为止，这个 ActionForm 实例一直存在，你可以引用同一个实例。

```
<forward name="create" path="/account/NewAccountForm.jsp" />
```

这表示 Action 执行完成后，需要推出的 Jsp 页面，name 值是你在 Action 执行子类中定义的名称；path 是你的 jsp 页面相对路径。

所以，编写一个传统意义上的 Jsp 页面，在 struts 变成下面几步：

1. 编写一个 ActionForm 实现子类
2. 编写一个 Action 实现子类，完成其中方法 execute 内容。
3. 配置 struts-config.xml
4. 使用 struts 标签编写 Jsp 页面，这时的 Jsp 页面和传统的 Jsp 相比：已经没有 Java 代码，完全 Html 语法。

由以上可见，struts 在实现设计优化目的同时，实际增加编程复杂性，Jdon 框架试图保证 struts 优点的前提下，简化标程复杂性，crud 的功能对上述步骤优化如下：

1. ActionForm 是 Model 代码复制；
2. 无需编写 Action 实现子类；
3. 模板化配置 struts-config.xml，参考别的配置，稍微修改即可。
4. 简化 Jsp 页面编写数量；四个功能只需编写一个 Jsp。

strtus 学习资源：http://www.jdon.com/idea/strutsapp/04005.htm

crud 设计起源

一般情况下，一个数据的增加修改的 Struts 实现流程如下：

（1）一般情况下，一个系统的操作用户（以下简称用户）新增或修改数据，首先要推送给他一个 Jsp 页面，如果是新增页面，就是一个空白表单的 Jsp 页面；如果是修改页面，则先到数据库中查询获得原来的数据，然后推出一个有数据表单的 Jsp 页面，用户才能在原来的数据上修改或编辑。

由于在 MVC 模式中，Jsp 页面只是一个页面输出，或者说不能有任何 Java 功能实现，因此上面修改页面推出前需要查询数据库这个需要 Java 实现的功能不能在 Jsp 页面中实现，只能在 Jsp 页面前加一个 Action，这样，修改页面的推出流程变为不是直接调用 Jsp 页面，而是：action.do ---> Jsp 页面，首先调用 Action；然后才由 Action 推出 Jsp 页面。

这个 Action 实现我们称为 ViewAction，专门用于控制输出 Jsp 界面，新增 Jsp 页面的推出前我们也加上这个 ViewAction，这样无论是新增 Jsp 页面或修改 Jsp 页面，都是由 ViewAction 推出，那么到底是推出新增 Jsp 页面还是修改 Jsp 页面呢？

关键是 ViewAction 的输入参数 Action 的值，根据 Action 的值来判断是新增还是修改。如果设置 Action 值如为空或为 create 值（如 http://xxx/viewAction.do?action=create），则直接输出新增性质的 JSP 页面；而 Action 值如为 edit（如 http://xxx/viewAction.do?action=edit），

则是要求输出进行编辑页面，根据 ID 查询数据库获得存在的数据，然后输出编辑修改性质的 JSP 页面。

当然，在 ViewAction 中还有一些具体参数的检查，如果是编辑，则关键字 ID 不能为空，因为后台要根据此 ID 为主键，查询数据库其相应的记录，如果数据库未查询到该主键的记录，则需要显示无此记录等信息。

（2）创建有关该数据的 JSP 页面，既用于新增页面，也用于修改页面。将该 Jsp 页面作为 ViewAction 的输出页面。该 Jsp 页面结构如下：

```
<html:form action=”/XXXSaveAction.do”>
    <html:hidden property="action" />
<!—该 action 的值是调用 viewAction.do?action 参数的值  -->
    …….
</html:form>
```

当用户填写完该 Jsp 页面中的表单数据将提交给一个新 Action 子类实现：专门用于接受表单数据并保存持久化它们。

（3） 创建 SaveAction，它用来接受提交表单的数据，不同于 ViewAction 专门用于输出 Jsp 表单页面，该 Action 专门用于接受 Jsp 表单页面提交的数据。

SaveAction 中主要是调用业务层 Service 实现数据持久化操作，调用 Service 之前，需要将表单的数据 ActionForm 转为一个对象（DTO 对象），然后作为方法参数传送给 Service 具体方法， Service 处理完成后，返回结果对象，SaveAction 还需要检查 Service 是否操作成功等。

总结：一个数据新增删除修改流程需要创建两个 Action，一个 Jsp 页面；当然爱 Struts 1.2 中已经通过 DispatchAction 解决了需要创建两个 Action 问题，只需要一个 Action，但是使用 Jdon 框架的 Struts 配置，一般情况下都不需要 Action，只要配置一下 JdonFramework.xml 如下即可：

```
<model key="typeId" class ="news.model.NewsType">
        <actionForm name="newsTypeActionForm"/>
        <handler><!--  以下相当于一个 action 代码实现  -->
          <service ref="newsManager">
            <getMethod name="getNewsType" />
            <createMethod name="createNewsType" />
            <updateMethod name="updateNewsType" />
            <deleteMethod name="deleteNewsType" />
          </service>
        </handler>
</model>
```

在以后章节将详细说明上面配置的语法。

crud 原则：一个数据表对应一个模型 Model；一个 Model 对应四个功能：新增；删除；修改和查询。

crud 设计重要角色

这里首先介绍一下 Jdon 框架在上面思路上延伸抽象设计的思路：

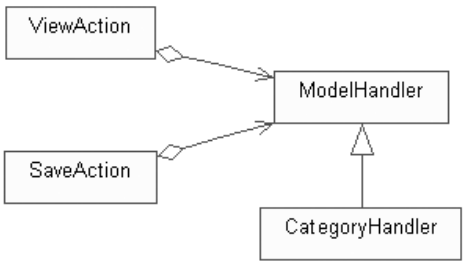
一个数据的增删改查流程有上面总结的流程组成：

ViewAction -→ 表单 Jsp 页面 -→ SaveAction --→ 结果 Jsp 页面

这样我们就可以将这个流程固定抽象到 Jdon 框架中，不同的是在其中流转的数据，既

然抽象出共性了，那么如何处理异性呢？

将这个流程中的异性使用继承实现，如下图：



ViewAction 和 SaveAction 中有关具体每个数据有不同的操作，都委托给 ModelHandler 实现，用户使用框架时，只要继承实现 ModelHandler，填补异性方法就可以。

以上是 Jdon 框架对 crud 流程的功能类设计规划，那么对于数据的抽象是如何设计的？

Jdon 框架以 com.jdon.controller.model.Model 抽象代表数据模型，在简单系统中，Model 又作为数据传送对象（DTO），在表现层和服务层以及持久层之间来回传递，由于表现层和持久层框架产品不同，Model 可能根据不同的表现层框架产品进行映射，如果表现层使用 Struts，那么显然 ActionForm 实际就是 Model 在表现层的映射或代名词，或者说是共同数据载体。

crud 规则一：Model 和 ModelForm

Model 是域模型，是采取领域模型分析法从系统需求分析中获得的，反映了应用系统的本质，Model 是一个简单的 POJO，属于数据类型的 POJO，区别于 POJO 服务。从传统意义上理解，Model 设计相当数据表设计，在传统的过程化编程中，一个数据库信息系统设计之前我们总是从数据表设计开始，而现在我们提倡是从域模型提炼开始。

Jdon 框架对模型对象建立有两个要求，继承 com.jdon.controller.model.Model，每个 Model 有一个主键。

每个 Model 需要一个主键

每个 Model 必须有一个主键，就象每个数据表设计都有主键一样，如果你的 Model 没有主键怎么办？那么使用强制给它一个 Object ID，这可以由一个专门序列器产生。

上述 Account 这个 Model 中主键是 username，这是依据 Account 对应的数据表 account 确定的，account 数据表设计中，username 是主键。

Model 配置

Model 在 jdonframework.xml 中配置，如下：

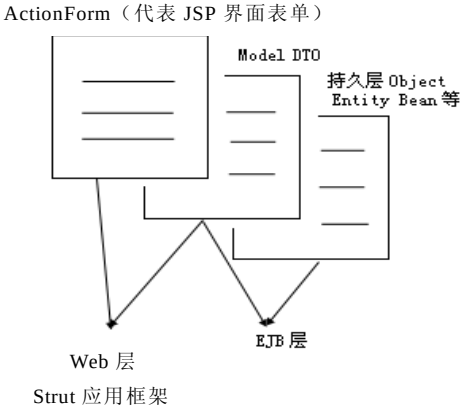
```
<model key="username" class ="com.jdon.framework.samples.jpetstore.domain.Account">
        …..
</model>
```

key 的值就是指定 Model 的主键，这里是 username，class 是 Account 的类，这样就定义了一个 Model。

Model 配置除定义 Model 自身属性以外，还需要定义了子属性：actionForm 和 handler，这两个定义在下面章节描述。

ModelForm 是 Model 的映射

Jdon 框架通过映射设计，保证表现层、Jdon 框架、服务层和持久层之间能够解耦独立，相互可插拔、脱离或组合。例如下图展示了表现层使用 Struts；服务层/持久层使用了 EJB 的映射设计：



Model 和 ModelForm 映射是通过相同字段拷贝实现的，也就是这两个类之间有相同的字段属性，那么字段属性的值可以在他们之间拷贝转移。例如 Model Account 的代码：

```
public class Account extends Model {

   private String username;
   private String password;
   private String email;
   private String firstName;
   private String lastName;
   ……

}
```

那么其 ModelForm 的类代码如下：

```
public class AccountForm extends ModelForm {

   private String username;
   private String password;
   private String email;
   private String firstName;
   private String lastName;
   ……

}
```

这样两者代码保证它们之间的映射，这是使用 Jdon 框架的规则之一。

ModelForm 配置

ModelForm 就是 ActionForm，因此只需要在 struts-config.xml 的<form-beans>中定义 ActionForm 就可以，如下：

```
<struts-config>
  <form-beans>
    <form-bean name="accountFrom"
                type="com.jdon.framework.samples.jpetstore.presentation.form.AccountForm"/>
  </form-beans>
</struts-config>
```

注意 ModelForm 的名字是 accountFrom，因为 Model 和 ModelForm 是映射对应关系，我们需要告诉 Jdon 框架这种对应关系。

那么，拓展前面的 Model 配置，在 jdonframework.xml 配置如下：

```
<model key="username" class ="com.jdon.framework.samples.jpetstore.domain.Account">
        <actionForm name="accountForm"/>
        ……
</model>
```

ModelViewAction 原理

Jdon 框架中的 com.jdon.strutsutil.ModelViewAction 是实现前面 ViewAction 原理的类，主要是实现在推出让用户新增数据或编辑数据的 Jsp 页面的之前的准备工作。它主要是做 Jsp 页面推出前的准备工作，然后推出不同的 Jsp 页面。

ModelViewAction 根据调用参数 action 的值，判断用户发出的是新增或编辑命令。

Action 有两个规定值 create 或 edit，也就是说，只有三种 URL 调用形式：

新增URL： http://xxxx/XXXModelViewAction.do?action=create

新增URL： http://xxxx/XXXModelViewAction.do

编辑URL： http://xxxx/XXXModelViewAction.do?action=edit

如果用户在浏览器发出这三种 URL 调用，将激活 ModelViewAction，ModelViewAction 将根据 action 的值分别推出用于创建性或编辑性的页面。下面分别详细说明这两种流程：

1.在创建性 Jsp 页面推出之前，要做一些准备工作，由于 Jsp 页面中表单数据（<form></form>之间数据）是由 ModelForm（ActionForm）概括的，所以，创建性 Jsp 页面推出前的准备工作主要是 ModelForm 的初始化创建，ModelForm 是 Struts 的 ActionForm 继承者，ModelForm 是抽象页面表单数据，它是 Model 在表现层的映射影子。

ModelForm 的初始化创建很重要，它决定了推出给用户创建性页面的内容，需要程序员介入度很高， Jdon 框架提供有几种初始化 ModelForm 方法：

首先检查 ModelHanlder 的 initForm 方法，检查用户有无自己实现初始化 ModelForm 实现；如果无，则用 Struts 的自动创建 ActionForm（ModelForm），这时的 ModelForm 实例是空对象，这时还有另外一种方式提供用户初始化 ModelForm 实例的方法：调用 Modelhandler 的 initModel 方法，将该方法返回的 Model 对象拷贝到 ModelForm 中，这符合 Model 和 ModelForm 相互映射的关系。下面章节将描述 ModelHandler 的 initForm 和 initModel 方法有何区别。

2.推出编辑性页面之前的准备工作主要是：根据主键查询从服务层获得一个已经存在的数据模型，简单地说：根据主键从数据库查询已经存在的一个记录，这样，推出的 Jsp 页面中包括数据的表单，用户可修改编辑。

这个工作涉及两部分：ModelForm 创建；然后从服务层获得一个有数据的 Model，将 Model 数据拷贝到 ModelForm 实例中。

crud 规则二：配置 ModelViewAction

根据不同域模型，我们有相应的不同的配置，例如：有一个 Model 为 A，那么为了推出 A 新增或修改或删除的页面，我们需要在 struts-config.xml 配置如下：

```
<action name="aActionForm" path="/aAction"    type="com.jdon.strutsutil.ModelViewAction">
  <forward name="create" path="/a.jsp" />
  <forward name="edit" path="/a.jsp" />
</action>
```

这是一个标准的 Action 配置，A 的 ActionForm 为 aActionForm，新增修改删除 A 的 Jsp 页面为 a.jsp，如上面配置，这样，如果用户从浏览器网址如下调用：

http://xxxxx/aAction.do?action=create

这样调用将推出 a.jsp 用于新增；

http://xxxxx/aAction.do?action=edit

这样调用将推出 a.jsp 用于修改删除。

再举例，如果 Model 为 B，B 的 ActionForm 为 bActionForm，Jsp 页面为 b.jsp，那么 Struts-config.xml 只要如下配置：

```
<action name="bActionForm" path="/bAction" type="com.jdon.strutsutil.ModelViewAction">
  <forward name="create" path="/b.jsp" />
  <forward name="edit" path="/b.jsp" />
</action>
```

如果 Model 为 c,那么 struts-config.xml 配置如下：

```
<action name="cActionForm" path="/cAction" type="com.jdon.strutsutil.ModelViewAction">
  <forward name="create" path="/c.jsp" />
  <forward name="edit" path="/c.jsp" />
</action>
```

总结上面配置，配置具备模板化编程的特点，拷贝粘贴然后修改，修改且有规律。

ModelSaveAction 原理

ModelSaveAction 是前面的 SaveAction 实现，它是真正的核心 Action 实现，专门将用户新增或修改后的数据提交到服务层，实现持久化如保存到数据库中。

ModelSaveAction 主要是委托 ModelHandler 实现数据提交和服务激活，这个过程比较单一有规律，用户可介入程度低，因此可以使用配置来代替代码实现。

我们看看 ModelSaveAction 是如何将 Jsp 页面提交的数据传递给后台，ModelSaveAction 从 ModelForm（ActionForm）中获得用户提交的数据，将其拷贝到相应的 Model 实例，然后将 Model 对象打包到 EventModel 对象中，将 EventModel 作为方法参数，调用 ModelHandler 的 serviceAction 方法，ModelHandler 的 serviceAction 则是调用服务层相应 Service 的对应的 create/update/delete 方法，所以 ModelSaveAction 主要工作委托给 ModelHandler 实现，我们将在下面研究重要的 ModelHandler 类。

crud 规则三：配置 ModelSaveAction

ModeSaveAction 和 ModelViewAction 类似，根据 Model 不同，在 struts-config.xml 中配置如下：

```
<action name="aActionForm" path="/aSaveAction" type="com.jdon.strutsutil.ModelSaveAction">
  <forward name="success" path="/aResult.jsp" />
  <forward name="failure" path="/aResult.jsp" />
</action>
```

如果 Model 为 B，那么 struts-config.xml 的配置是：

```
<action name="bActionForm" path="/bSaveAction" type="com.jdon.strutsutil.ModelSaveAction">
  <forward name="success" path="/bResult.jsp" />
  <forward name="failure" path="/bResult.jsp" />
</action>
```

关于 ModeSaveAction 使用是在 Jsp 页面中赋值给 action：

```
<html:form action="/aSaveAction.do">
  <html:hidden property="action" />
  <html:text ….
  ….
</html:form>
```

该 Jsp 页面是新增或修改性质的页面，例如可以是前面介绍的 ModelViewAction 推出的 a.jsp 或 b.jsp。

至此，ModelViewAction -→ jsp →ModelSaveAction 实现了首尾衔接，实现了一个 crud 操作流程。

通过配置使用 Jdon 框架中的 ModelViewAction 和 ModelSaveAction，程序员避免了象开发普通 Struts 应用系统那样建立至少一个 Action 子类。

在这个流程中，需要和服务层服务交互，这需要由程序员根据具体程序定制，下面介绍程序员可介入定制的重要类 ModelHandler。

规则四：配置/代码实现 ModelHandler 类

ModelHandler 是前面 ModelViewAction 和 ModelSaveAction 的委托者，程序员需要定制实现的部分由继承 ModelHandler 实现。

目前，Jdon 框架提供两种 Modelhandler 的实现：编码和配置。

ModelHandler 几个重要方法是总结了表现层和服务层三种交互操作，如果你的应用不属于这三种交互操作，那么可能就无法使用 Jdon 框架的 crud 功能，直接使用 Struts 实现。

ModelHandler 有几个重要方法是 ：initForm 方法、initModel 方法、findModelByKey 方法和 serviceAction 方法。前面两个方法是和页面初始化有关，页面初始化有可能和服务层交互访问；findModelByKey 是根据主键查询数据库存在的记录，这是和编辑页面相关，编辑之前需要先查询，这个方法也需要和服务层交互访问；serviceAction 则是将用户对数据的新增修改删除决定通知服务层进一步处理，这是和服务层主要交互访问。

这几种主要交互操作详细描述如下：

规则四.一：页面初始化

推出页面需要初始化，推出新增性和编辑性页面的初始化工作是不一样的。我们知道，Jsp 页面的初始化其实是 ActionForm（ModelForm）的初始化，我们分别从新增和编辑页面的初始化两条线路说明。

如果你需要给 ModelForm 中属性字段简单初始化一些常量，那么很显然这些在 ModelForm 的构造方法中实现，如：

```
public class AccountForm extends ModelForm{
```

```
    private List languages;
    public AccountForm() {
      languages ＝ new ArrayList();
      languages.add("english");
      languages .add("japanese");
   }
}
```

上面代码需要给 languages 加入一些初始值，这样，推出页面时，用户可以进行多项选择（使用 html 的 select/options multibox 等实现）。

如果初始化这些属性值需要从数据库中获得，那么就需要和服务层实现交互了，那么就要考虑继承实现 ModelHandler 了。

ModelForm 创建可由通过服务层后台交互实现，这种交互实现也有两种方式：

1. 通过代码实现：ModelHandler 提供的 initForm 方法， initForm 程序员只要继承 ModelHandler，在 initForm 方法中调用服务层服务，获得初始化值，再赋值入 ModelForm，如下代码：

```
public class NewsHandler    extends ModelHandler {

   public ModelForm initForm(HttpServletRequest request) throws Exception {
      Debug.logVerbose("enter iniForm .", module);
      NewsActionForm nf = new NewsActionForm(); //创建 ModelForm
      NewsManagerLocal newsManager = (NewsManagerLocal)
                              WebAppUtil.getEJBService("newsManager",    request);
      PageIterator pi = newsManager.getNewsTypePage(0, 50);
      Collection newsTypes = new ArrayList();
      while (pi.hasNext()) {
         String id = (String) pi.next();
         NewsType newsType = newsManager.getNewsType(id);
         newsTypes.add(newsType);
      }
      nf.setNewsTypes(newsTypes); //将新闻类型列表赋值到新闻这个 ModelForm
      return nf;
   }
   ……
}
```

还需要配置一下，告诉 Jdon 框架你的 ModelHandler 实现：

```
<model ……>
        <handler class="news.web.NewsHandler" />
</model>
```

以上是代码实现直接初始化 ModelForm，还有一种初始化 ModelForm，思路是：初始化 Model，然后将 Model 值拷贝到 Model 中，这是通过 ModelHandler 的 initModel 实现的。不过 initModel 和 initForm 只能选择其一实现，initModel 还可以通过配置实现：

2. 通过配置实现，配置主要是通过 initModel 方法实现，在 jdonframework.xml 中配置如下：

```
<model ….>
        <handler>
          <service ref="accountService">
            <initMethod     name="initAccount" />
```

```
            …..
          </service>
        </handler>
</model>
```

上面配置 initMethod 方法是访问 accountService 的 initAccount 方法，也就是说，ModelForm 的初始化推给 Model 初始化，而 Model 初始化则由 accountService 的 initAccount 实现，程序员必须实现的 accountService 的 initAccount 方法。如：

```
public class AccountServiceImp implements AccountService{
    private ProductManager productManager;
    public AccountServiceImp(ProductManager productManager){
        this.productManager = productManager;
    }

public Account initAccount(){
        Account account = new Account();
        account.setCategories(productManager.getCategoryList());
        return account;
    }
    ….
```

这两种页面初始化选用依据，是根据用来初始化的数据来自何处？是来自后台或服务层，那么选择第二个方案；如果来自 request 相关的例如 HttpSession，则选用第一个方案。

规则四.二：编辑之前的数据查询

前面是实现页面初始化，对于推出的编辑性页面，查询获得已经存在的数据也属于一种初始化工作，这是通过 ModelHandler 的 findModelByKey 实现的，这也有两种实现方法：

1. 代码实现，继承 ModelHandler 实现，如果你的 Model 查询不是从数据库获得，如是从 HttpSession 获得的，那么你要继承实现自己的方法 findModelByKey。代码实现后，记住配置 jdonframework.xml：

```
<model ……j>
    <handler class="你的 ModelHandler 子类" />
</model>
```

一般情况下，是通过服务层查询数据库获得的，那么涉及到服务层交互，可以采取配置实现。

2. 配置实现，在 jdonframework.xml 配置如下：

```
<model ….>
    <handler>
      <service ref="accountService">
          <getMethod name="getAccount" />
        …..
      </service>
    </handler>
</model>
```

那么，要求你的服务层服务有相应的方法如下：

```
public class AccountServiceImp implements AccountService{
    private ProductManager productManager;
    public AccountServiceImp(ProductManager productManager){
        this.productManager = productManager;
```

```
        }

        public Account getAccount(String username) {
            Account account = null;
            try{
                account = accountDao.getAccount(username);
                account.setCategories(productManager.getCategoryList());
                //other business logic
            }catch(DaoException daoe){
                Debug.logError(" Dao error : " + daoe, module);
            }
            return account;
        }
        ….
    }
```

注意，在此配置中，对服务层的服务方法有一个规定：getAccount 的方法参数类型必须是 String 型的，getAccount 返回必须是 Model 类型，上面 Account 类是继承 Model 类的。

规则四.三：数据提交保存

用户在 Jsp 页面填写或修改数据后，将实现保存提交，或实现删除提交，这时 Jsp 页面的 action 是提交给 ModelSaveAction 的，ModelSaveAction 委托 ModelHandler 的 serviceAction 实现结果保存或删除。serviceAction 有两种实现：

1.代码实现，这部分工作主要是实现传递工作，代码比较有规律，一般使用配置实现，如果你需要在提交之前实现一些其他特殊实现，那么使用代码实现如下：

```
    public void serviceAction(EventModel em, HttpServletRequest request) throws
            java.lang.Exception {
        try { //从 Httpsession 获得 User，保存到 News 中
            User user = (User) ContainerUtil.getUserModelAfterLogin(request);
            if (user != null) {
                News news = (News) em.getModel();
                news.setUser(user);
            }

        NewsManagerLocal newsManager = (NewsManagerLocal) WebAppUtil.
                getEJBService("newsManager",   request);

            switch (em.getActionType()) {
                case Event.CREATE:
                    newsManager.createNews(em); //递交 service 创建方法
                    break;
                case Event.EDIT:
                    newsManager.updateNews(em);//递交 service 修改方法
                    break;
                case Event.DELETE:
                    newsManager.deleteNews(em);//递交 service 删除方法
                    break;
            }
        }
```

```
        catch (Exception ex) {
            throw new Exception(" serviceAction Error:" + ex);
        }
    }
```

还需要配置 jdonframework.xml 你的 ModelHandler 代码实现：

```
<model ……j>
        <handler class="news.web.NewsHandler" />
</model>
```

2.配置实现，如果在递交服务处理之前没有特殊工作实现，可使用配置，如下：

```
<model ….>
        <handler>
          <service ref="accountService">
                    <createMethod name="insertAccount" />
            <updateMethod name="updateAccount" />
                    <deleteMethod name="deleteAccount" />
            …..
          </service>
        </handler>
</model>
```

那么，要求你的 Service 有 insertAccount 和 updateAccount 或 deleteAccount 方法，而且这三个方法参数必须是 EventModel 类型，当然，方法名是可以根据你的 service 具体方法不同，在配置中更改，例如你的 Service 有 insert 方法，那么配置就是：

<createMethod name="insert" />

方法名可以变化，任意取，但是方法参数必须是 EventModel 类型的。如：

```
public interface AccountService {
      Account initAccount();   //初始化
       Account getAccount(String username); //查询获得存在的 Model
       void insertAccount(EventModel em); //新增方法
       void updateAccount(EventModel em); //修改方法
}
```

当然，如果你使用代码实现，就没有这些对 Service 类的编程规定了。

这里稍微谈一下上面配置实现代码功能原理：上面配置在 Jdon 框架中是自动调用 com.jdon.model.handler.XmlModelhandler 作为代码实现，也就是说，XmlModelhandler 根据上面配置自动生成前面的代码，属于一种简单的代码自动生成。

注意：有时我们可能需要代码配置混合实现，ModelHandler 几个方法中只代码实现一个方法，其他都可以配置实现，这也是可以的，只是这时你的代码不是继承 ModelHandler，而是 XmlModelHandler 了，同时配置要写明你的子类实现：

```
<model ….>
        <handler class=”你的 XmlModelHandler 子类”>
          <service ref="accountService">
                    <createMethod name="insertAccount" />
            <updateMethod name="updateAccount" />
                    <deleteMethod name="deleteAccount" />
            …..
          </service>
        </handler>
</model>
```

struts-config.xml 配置技巧

crud 功能实现标准配置

参考前面的ModelViewAction和ModelSaveAction配置章节，一个数据Model的标准crud功能实现无需编写任何 Action，只要有 ActionForm（还是 Model 的影子），通过struts-config.xml 配置就可以实现：

1.推出创建性或编辑性页面：

```
<action name="aActionForm" path="/aAction" type="com.jdon.strutsutil.ModelViewAction">
  <forward name="create" path="/a.jsp" />
  <forward name="edit" path="/a.jsp" />
</action>
```

约束：

（1）type 必须是 com.jdon.strutsutil.ModelViewAction

（2）forward 的 name 值必须是 create 或 edit

2.接受数据提交数据并递交 Service 服务层处理。

```
<action name="aActionForm" path="/aSaveAction" type="com.jdon.strutsutil.ModelSaveAction">
  <forward name="success" path="/aResult.jsp" />
  <forward name="failure" path="/aResult.jsp" />
</action>
```

约束：

type 必须是 com.jdon.strutsutil.ModelSaveAction

forward 的 name 值必须是 success 或 failure

crud 功能实现分离配置

crud 可以通过两行 action 配置 ModelViewAction ModelSaveAction 合并完成，也可以将新增和修改分开配置，例如用户注册功能，用户注册（用户创建）和用户资料修改属于不同的需要分离开的两个过程，用户注册功能是针对新用户，或者说是非注册用户；而用户资料修改是针对注册用户，两者服务对象角色是不一样的，因此，不能象其他 Model 那样将 crud 合并在一起。

1. 推出创建性页面的配置：

```
<action name="accountForm" path="/shop/newAccountForm"
        type="com.jdon.strutsutil.ModelViewAction" scope="session">
  <forward name="create" path="/account/NewAccountForm.jsp" />
</action>
```

将 forward 的 edit 值为空，没有这一行。

2. 接受创建数据并并递交 Service 服务层处理

```
<action name="accountForm" path="/shop/newAccount"
        type="com.jdon.strutsutil.ModelSaveAction" scope="session"
       validate="true" input="/account/NewAccountForm.jsp">
  <forward name="success" path="/shop/index.shtml" />
  <forward name="failure" path="/shop/newAccountForm.shtml" />
</action>
```

3. 推出编辑性页面的配置

```
<action name="accountForm" path="/shop/editAccountForm"
```

```
            type="com.jdon.strutsutil.ModelViewAction" scope="session">
        <forward name="edit" path="/account/EditAccountForm.jsp" />
    </action>
```

将 forward 的 create 值为空，没有这一行，不过调用/shop/editAccountForm.shtml 形式还是必须要有参数的：/shop/editAccountForm.shtml?action=edit&username=XXX

4. 接受修改后的数据并并递交 Service 服务层处理

与第 2 条类似，不同的是 path 和 jsp 因为编辑页面不一样而不同：

```
    <action name="accountForm" path="/shop/editAccount"
            type="com.jdon.strutsutil.ModelSaveAction" scope="session"
            validate="true" input="/account/EditAccountForm.jsp">
        <forward name="failure" path="/shop/editAccountForm.shtml" />
        <forward name="success" path="/shop/index.shtml" />
    </action>
```

单纯输出 Jsp 页面

如果有时只是为了输出一个 Jsp 页面而做一个 Action 比较麻烦，Jdon 框架实现了一个缺省的转发 Action，可以简单重用在很多这样场合。只要在 struts-config.xml 配置如下：

```
<action path="/XXX" type="com.jdon.strutsutil.ForwardAction"
        name="XXX" scope="request"          validate="false">
        <forward name="forward" path="/xxx.jsp"/>
</action>
```

上述配置有三点要求如下：

type 必须是 com.jdon.strutsutil.ForwardAction

forward 的 name 值是 forward

Jdon 框架的出错信息

在服务层一旦实现数据库操作出错，Jdon 框架通过 EventModel 的 setErrors 方法向表现层传递出错信息，因此，只要在你的服务层 Service 实现方法的出错代码调用该方法即可，如：

```
public void insertOrder(EventModel em) {
        Order order = (Order)em.getModel();
        try{
            orderDao.insertOrder(order);
        }catch(Exception daoe){
            Debug.logError(" Dao error : " + daoe, module);
            em.setErrors("db.error");
        }
    }
```

当存储 orderDao 调用出错，将 db.error 保存到 EventModel 中，而 db.error 是 struts 的 Application.properties 中定义的,，需要在你的应用系统中定义，Jdon 框架应用到的信息如下：

```
id.required = you must input id
id.notfound = sorry, not found
db.error=sorry, database operation failure!
```

```
system.error=sorry, the operation failure cause of the system errors.
maxLengthExceeded=The maximum upload length has been exceeded by the client.
notImage=this is not Image.
```

主要有 id.required id.notfound db.error 等几个，需要在 J2EE 应用系统中定义。

批量分页查询快速开发

原理

批量查询由于是频繁操作，对系统性能设计要求很高，而且批量查询几乎存在每个数据库信息系统，因此，这个功能具有可重用性，Jdon 框架根据 Jive 等传统成熟系统批量查询设计原理，总结多个应用系统，在不破坏多层情况下，抽象出批量查询子框架。

批量查询设计两个宗旨：

1. 尽量减少数据库访问，减少数据库负载和 I/O 性能损耗。
2. 由于 J2EE 是一个多层结构，尽量减少在多个层次之间传送的数据量。减少传送损耗。

因此，批量查询设计将涉及到两个层面：表现层和持久层，Jdon 框架提供了持久层下 JDBC 查询操作模板（JdbcDao），这个 JdbcDao 可以被服务层不同服务类型（POJO Service 或 Session Bean）调用。

根据批量查询设计宗旨，有下面实现要点：

1. 使用缓存减少数据库访问
2. 持久层不能将满足查询条件的每页所有完整数据记录传送到表现层，试想一下，如果一条数据记录有 1000 个字节，那么一页显示 20 条，那么就有 2 万个字节传送，目前很多批量查询设计都是这样做，存在浪费内存和性能消耗大的隐患。
3. 缓存越靠近用户界面端，性能越好，因此，持久层如果只传送满足查询条件的数据记录主键（一个字段）集合，那么表现层可以优先从缓存中根据主键获得该数据完整数据，查询越频繁使用，缓存击中率越高，各方面消耗就越小。

根据以上设计原则，Jdon 框架的批量查询设计方案如下：

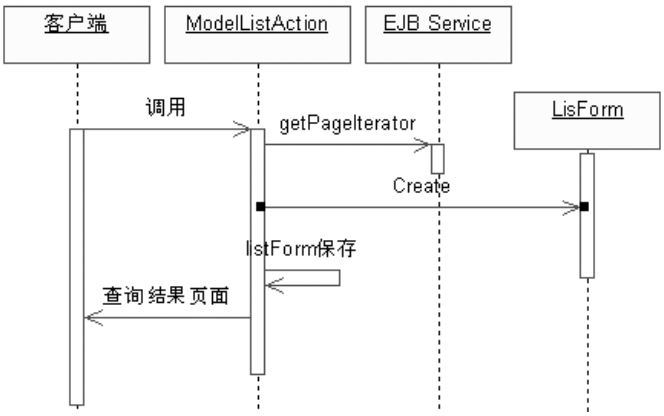
对于持久层：

- 获取满足查询条件的所有数据表记录总个数。
- 获取当前页面所有数据记录主键集合（ID 集合）。
- 将上述两种数据打包成一个对象（PageIterator）传送到前台

对于表现层：

- 获得后台满足查询条件的 PageIterator 对象。
- 遍历 PageIterator 对象中的主键集合，根据主键再查询后台获取完整数据对象（Model），先查询缓冲，如果没有，再访问数据库。
- 将获取的完整数据对象（Model）封装成一个集合，打包成 ModelListForm 对象。
- Jsp 页面再展开 ModelListForm 对象中 Model 数据集合，逐条显示出来。

批量查询的主要示意图：



图中解释：后台将满足查询条件的数据记录的主键集合（ID 集合）包装成 PageIterator 对象，然后由服务层返回到表现层（通过 ModelListAction 调用服务 EJBService 或 POJO Service 的 getPageIterator 方法），表现层的 ModelListAction 再遍历这个 ID 集合，根据每个 ID 查询获得完整 Model 数据，再打包到 ModelListForm 对象中。

批量查询框架基于一个前提是：每个 Model 有一个主键，就象每个数据表设计都有主键一样，如果你的 Model 没有主键怎么办？那么使用强制给它一个 Object ID，这可以由一个专门序列器产生。这与前面 crud 实现的前提是一致的。

重要对象 PageIterator

从前面批量查询原理中看出，PageIterator 实际是一个在多层之间穿梭的数据载体，前后台关系只是靠 PageIterator 来实现消息交互。

因此，无论后台持久层使用什么技术（JDBC、实体 Bean、Hibernate 或 iBatis），只要能提供一个查询结果对象 PageIterator 给前台就可以。

考虑到持久层技术发展极快，Jdon 框架并没有在持久层的缺省实现，但提供了 JDBC 实现的帮助 API（见包：com.jdon.model.query），具体可参考 Jdon 框架源码包 Samples 目录下 JdonNews 的 news.ejb.dao. JdbcDao。这个 JdbcDao 虽然为 EJB Session Bean 调用，也可以为普通的 POJO 服务调用，JdbcDao 开发也有模板化，一般是一个 Model 完成三个方法，具体可见“在 JBuilder 下开发 Struts+Jdon+EJB”章节的“批量查询持久层实现”。

在 Jdon-Jpetstore 中，批量查询持久层实现是使用 iBatis，然后在服务层将 iBatis 的 PaginatedList 类转换成 Jdon 框架的 PageIterator 即可。

com.jdon.controller.model.PageIterator 类代码如下：

```
public class PageIterator implements Iterator, Serializable {

    public final static Object[] EMPTY = new Object[0];

    private int allCount = 0; //符合查询条件的所有 Model 总数
    private Object[] keys; //符合查询条件的当前页面的 Model 的 ID 集合
    private int currentIndex = -1; //遍历当前页面时的记录指针
    private Object nextElement = null; //下一个记录

    private int start;   //当前页面在所有符合查询条件中开始点
    private boolean hasNext;   //是否有下一页
    //以上两个参数是有关页为单位的信息
```

```
    /**
     * 完整的构造器
     * allCount 是 PageIterator 重要参数
     */
    public PageIterator(int allCount, Object[] keys, int start, boolean hasNext) {
        this.allCount = allCount;
        this.keys = keys;
        this.start = start;
        this.hasNext = hasNext;
    }

    /**
     * 当前页面的构造器
     * allCount 是 PageIterator 重要参数，也需要赋值，
     * 因为其值对于每个页面是一样的，所以可使用本构造方法构造后，再从缓存
     * 中读取 allCount， 使用 setAllCount 延迟赋值
     */
    public PageIterator(Object[] keys, int start, boolean hasNext) {
        this.allCount = 0;
        this.keys = keys;
        this.start = start;
        this.hasNext = hasNext;
    }

    /**
     * 空构造器
     * 使用本构造器可防止没有符合查询条件时，Jsp 页面报出 nullException 讨厌错误。
     */
    public PageIterator() {
            this.allCount = 0;
            this.keys = EMPTY;
            this.start = 0;
            this.hasNext = false;
        }
      …..
}
```

持久层参考 API：PageIteratorSolver

Jdon 框架提供了 PageIterator 创建的持久层创建 API：com.jdon.model.query.PageIteratorSolver，这是一个基于 SQL 的 JDBC 实现，当然你完全可以其他持久层技术实现创建 PageIterator。

使用 Jdon 框架的 PageIteratorSolver 提供 JDBC 模板操作，在查询功能上几乎无需写 JDBC 语句。使用 PageIteratorSolver 可以书写很少代码就实现一种 Model 的批量分页查询功能，PageIteratorSolver 可以被 Session Bean 调用，或者作为普通 POJO 被调用，缺省情况下 PageIteratorSolver 不会配置在 Jdon 框架容器中。

PageIteratorSolver 创建有两种方式：

第一种：直接 new，PageIteratorSolver 提供两种构造器，具体参考其 API，一种构造器可以更换缓存器，另外一种是缺省的；这两种构造器都需要 DataSource 作为构造参数。如下：

```
ServiceLocator sl = new ServiceLocator(); //Jdon 框架中的 ServiceLocator
DataSource    dataSource = (DataSource) sl.getDataSource("java:/NewsDS");
PageIteratorSolver    pageIteratorSolverOfType = new PageIteratorSolver(dataSource);
```

因为 PageIteratorSolver 中包含缓存器，因此，你可以为每个 Model 建立一个对应的 PageIteratorSolver 对象，这样，该 Model 更新时，只要刷新这个 Model 相关的缓存，具体可见下面缓存清除方法。

第二种，将 PageIteratorSolver 作为 POJO Service 配置在 jdonframework,xml，然后通过组件实例方式获得（如果它的调用者是以组件服务方式获得）。

创建 PageIterator

首先，根据批量查询原理，首先要查询符合查询条件的记录总数和当前页面的所有 Model 的 ID 集合，这是通过 PageIterator 的创建实现。

PageIteratorSolver 中重要方法 getPageIterator 是为了创建一个 PageIterator 对象，需要的输入参数相当比较复杂，这里详细说明一下：

getPageIterator 方法如下：

```
public PageIterator getPageIterator(String sqlqueryAllCount,
                String sqlquery, String queryParam, int start, int count) throws Exception
```

注意：PageIteratorSolver 的 getDatas 与 getPageIterator 实则一样，getPageIterator 名称或参数都显得易懂正规一些。

```
public PageIterator getDatas(String queryParam, String sqlqueryAllCount,
                                        String sqlquery, int start,
                                        int count) throws Exception
```

首先从 String sqlqueryAllCount 参数解释:

sqlqueryAllCount 参数其实是一个 sql 语句，该 sql 语句用来实现查询满足条件的所有记录的总数。例如查询某个表 myTable 的 sql 语句如下：

```
String    sqlqueryAllCount    =    "select count(1) from myTable ";
```

当然这是一个标准 sql 语句，后面可写入 where 等查询条件

sqlquery 参数则是关键的查询语句，主要实现批量查询中查询符合条件的数据记录（Model）的主键 ID 集合。例如表 myTable 的主键是 id，则

```
String sqlquery = select    id    from T_ myTable where categoryId = ?
```

queryParam 参数则是和 sqlquery 参数相关，如果 sqlquery 语句中有查询条件“?”，如上句 categoryId = ?中?的值就是 queryParam 参数，这样 sqlquery 和 queryParam 联合起来形成一个查询语句。如果没有查询条件，那么 queryParam 就赋值为 null 或""。

注意，如果查询条件有多个“?”，那么就需要将这些 queryParam 打包成一个 Collection，然后调用方法：

```
public PageIterator getPageIterator(String sqlqueryAllCount,
                String sqlquery, Collection queryParams, int start, int count) throws Exception
```

这个方法与前面的 getDatas 区别就是方法参数由 String queryParam 变成集合 queryParams，queryParams 是多个查询条件“?”的集合，注意，查询条件目前 Jdon 框架支

持常用的几种查询参数类型：String Integer Float 或 Long，例如，如果你的查询条件如下：

```
String sqlquery = select    id    from T_ myTable where categoryId = ? and name = ?
```

这里有两个查询参数，那么将这两个查询参数打包到数组中即可，注意顺序和问号的顺序是一样，如果问号对应的变量 categoryIdValue 和 nameValue，则如下：

```
Collection queryParams = new ArrayList();
queryParams.add(categoryIdValue);
queryParams.add(nameValue);
```

参数 start 和 count 则是有表现层传入的，start 表示当前页面是从所有符合查询条件记录中第几个开始的，而 count 则是从 start 开始查询多少条记录个数，也就是一页显示的记录数目。

单个 Model 查询

前面已经获得 PageIterator 对象，Jdon 框架会在 ModelListAction 中遍历这个 PageIterator 对象中 ID 集合，然后根据 ID 先从缓存中获取完整 Model 数据，如果没有则从数据库获得，那么我们在持久层还需要提供单个 Model 查询的实现。

Jdon 框架已经内置查询模板，无需使用者自己实现 JDBC 语句操作，直接使用模板即可，以 JdonNews 中 Model ： NewsType 查询一个实例为例子：

首先确定 sql 语句写法，如下：

```
String GET_TYPE = "select    * from T_NEWS_TYPE where typeid = ?";
```

其中 typeid 参数值由外界传入，因此写方法如下：

```
  public NewsType getNewsType(String Id) throws Exception {
     String GET_TYPE = "select    * from T_NEWS_TYPE where typeid = ?";
     List queryParams = new ArrayList();
     queryParams.add(Id); //Id 是 GET_TYPE sql 语句中的?的值


     …..
}
```

上述代码准备了 JDBC 模板查询两个参数：GET_TYPE 和 queryParams，

调用 PageIteratorSolver 中的两个 Model 查询方法：

```
  //适合查询返回结果是单个字段，如：
  // select name from user where id=?
  public Object querySingleObject(Collection queryParams, String sqlquery)    throws Exception ;

  //适合查询返回结果是多个字段，而且有多项记录
  //例如：select id, name, password from user where id = ?
  public List queryMultiObject(Collection queryParams, String sqlquery)    throws Exception ;
```

这两个方法适合不同的查询语句，当你的查询 sql 语句返回不只一条数据记录，而且每条数据记录不只是一个字段，而是多个字段，使用 queryMultiObject，queryMultiObject 经常使用。下面是获得一个 Model 查询的完整写法：

```
  public NewsType getNewsType(String Id) throws Exception {
     String GET_TYPE = "select    * from T_NEWS_TYPE where typeid = ?";
     List queryParams = new ArrayList();
     queryParams.add(Id);

     NewsType ret = null;
     try {
```

```
            List list = pageIteratorSolverOfType.queryMultiObject(queryParams, GET_TYPE);
            Iterator iter = list.iterator();
            if (iter.hasNext()) {//遍历查询结果，根据你的判断，决定这里使用 if 或 while
               Map map = (Map)iter.next(); //List 中每个对象是一个 Map
               ret = new NewsType();
               ret.setTypeName((String)map.get("typename")); //根据字段名称从 Map 中获取其相应值
               ret.setTypeId(Id);
            }
         }
         catch (Exception se) {
            throw new Exception("SQLException: " + se.getMessage());
         }
         return ret;
      }
```

缓存清除

由于 PageIteratorSolver 内置了缓存，缓存两种情况：

1. 符合查询条件的所有记录总数，这个总数第一次使用后将被缓存，不必每次都执行 select count 的数据操作，这是很耗费数据性能的。
2. 当前页面中所有 Model 的 ID 集合，在没有新的 Model 被新增或删除情况下，这个 ID 集合几乎是不变的，因此其结果在第一次被使用后将被缓存。

但是，某个 Model 发生新增或删除情况下，我们要主要清除这些缓存，否则新增或删除的 Model 将不会出现在 Model 列表中，PageIteratorSolver 缓存设计前提是 Model 的新增和删除不会很频繁，至少没有查询频繁。

PageIteratorSolver 的 clearCache()提供主动缓存清除。

我们需要在 Model 的新增方法或删除方法中，在这些方法成功后，主动调用 PageIteratorSolver 的 clearCache 方法。

因为 Model 的新增或删除方法是由程序员自己实现，如使用 CMP 或 Hibernate 等实现，因此需要注意加入 clearCache 方法。

我们建议 PageIteratorSolver 创建依据每个 Model 有一个对应的 PageIteratorSolver 对象，这样，这个 Model 变动只会清除它的有关缓存；如果所有 Model 共用一个 PageIteratorSolver 对象，一旦一个 Model 变动将引起所有缓存清除，降低缓存效益。

其他形式 PageIterator 创建

上面是使用 Jdon 框架的 JDBC 模板实现 PageIterator 创建，你有可能使用其他技术实现持久层，这里提供两种 PageIterator 创建的参考代码：

例如，productDao 是封装了别的持久层技术(iBatis 或 Hibernate)。要求两步就可以完成 PageIterator 创建：

第一步. productDao 返回符合条件的当前页面 ID 集合：

```
List list = productDao.getProductIDsListByCategory(categoryId, start, count);
```

第二步. productDao 返回符合条件的总数：

```
int allCount = productDao.getProductIDsListByCategoryCount(categoryId);
```

创建 PageIterator 代码如下：

```
int currentCount = start + list.size();//计算到当前页面已经显示记录总数
```

```
PageIterator pageIterator = new PageIterator(allCount, list.toArray(), start,
                                (currentCount < allCount)?true:false);
```

以上代码是在服务层实现，一旦服务层能够返回一个 PageIterator 实例，就可以按照下面表现层实现完成批量分页查询功能。

另外一个 PageIterator 创建实现，假设所有数据 ID 或数据模型包装在一个 List 中，从 List 中创建 PageIterator 的代码如下：

```
//计算未显示记录个数
int offset = itemList.size() - start;
int pageCount = (count < offset)?count:offset;
//生成当前页面的 List
List pageList = new ArrayList(pageCount);
//从 start 开始遍历，遍历次数是一页显示个数
//将当前页面要显示的数据取出来放在 pageList 中
for(int i=start; i< pageCount + start;i++){
      pageList.add(itemList.get(i));
}
int allCount = itemList.size();
int currentCount = start + pageCount;
PageIterator pageIterator = new PageIterator(allCount, pageList.toArray(), start,
            (currentCount < allCount)?true:false);
```

表现层实现：

前面章节主要谈论了批量查询的持久层实现，Jdon 框架的批量查询主要在表现层封装了主要设计原理，让我们看看表现层 Struts 的实现步骤：

ModelListForm

这里以查询商品类别（Category）下所有商品(Product)列表为需求，演示批量的查询的开发步骤。

使用 Jdon 框架实现批量查询时，无需使用代码创建 ModelForm（ActionForm），Jdon 框架提供一个批量查询的缺省实现：com.jdon.strutsutil.ModelListForm

只要在 struts-config.xml 中配置这个 ActionForm 就可以：

```
<form-bean name="productListForm" type="com.jdon.strutsutil.ModelListForm"/>
```

习惯地，我们将这个 ActionForm 命名为：

model 名称+ListForm

例如，查询商品列表是显示一个商品 Prodcut 数据，那么这个 ActionForm 的名称就是 productListForm

ModelListForm 是一个普通的 JavaBeans，主要属性如下：

```
public class ModelListForm extends ActionForm{

   private int allCount = 0;   //符合查询条件的所有记录总数
   private int start = 0;     //当前页面的开始
   private int count = 20;    //当前页面的可显示的记录数
   private boolean hasNextPage = false;   //是否有下一页
```

```
    /**
     * 父 Model
     */
    private Model oneModel = null;

    /**
     * 批量显示的 Model 集合
     * 这是本类主要的属性
     */
    private Collection list = new ArrayList();

    ……
}
```

抽象类 ModelListAction

com.jdon.strutsutil.ModelListAction.ModelListAction 是一个 struts 的标准 Action，它主要实现如下功能：

1. 从服务层获得符合查询条件 PageIterator，这是需要通过 getPageIterator 方法实现；
2. 展开遍历 PageIterator，根据 ID，首先从缓存中查询是否有其 Model（完整数据记录），如无，则调用服务层从持久层数据库获得，这是需要通过 findModelByKey 方法实现。
3. 将获得的 Model 集合封装在 ModelListForm 的 **list** 字段中。

所以，ModelListAction 实际是将 ID 集合换算成 Model 集合，然后交给 ModelListForm，Jsp 页面的显示只和 ModelListForm 有关。

```
public abstract class ModelListAction extends Action {
    private ModelManager modelManager;

    //struts action 的缺省 execute 方法
    public ActionForward execute(ActionMapping actionMapping,
                                 ActionForm actionForm,
                                 HttpServletRequest request,
                                 HttpServletResponse response) throws Exception {
        ……

        //从服务层获得符合查询条件的 PageIterator
        PageIterator pageIterator = getPageIterator(request, start, count);
        if (pageIterator == null) {
            throw new Exception(
                    "getPageIterator's result is null, check your ModelListAction subclass");
        }
        //获得 ModelListForm 实例
        ModelListForm listForm = getModelListForm(actionMapping, actionForm,
                                                  request, pageIterator);
        //赋值页面起始数等值到 ModelListForm
        listForm.setStart(start);
        listForm.setCount(count);
```

```
        listForm.setAllCount(pageIterator.getAllCount());
        listForm.setHasNextPage(pageIterator.isNextPageAvailable());

        //根据 pageIterator 获得 Model 集合
        Collection c = getModelList(request, pageIterator);
        Debug.logVerbose(" listForm 's property: getList size is " + c.size(), module);
        //将 Model 集合赋值到 ModelListForm 的 list 字段
        listForm.setList(c);
        //设置其他 Model 到 ModelListForm，以便 jsp 页面能显示其他 Model
        listForm.setOneModel(setupOneModel(request));
        //程序员自己定义的优化 ModelListForm 其他动作，供实现者自己优化。
        customizeListForm(actionMapping, actionForm, request, listForm);

        ……
    }
```

ModelListAction 是一个抽象类，它必须有一个实现子类需要有两个方法必须实现：

1. 获得服务层的 Pageiterator，也就是 getPageIterator 方法
2. 根据 key 或 ID 获得单个 Model 的方法，findModelByKey 方法。

其他可选实现的方法有：

1. 是否激活缓存方法

   protected boolean isEnableCache()

   有时，批量查询实现可能不需要缓存，获得每个 Model 都一定要执行 findModelByKey 方法。通过覆盖 isEnableCache 方法实现。
2. 批量查询的 Jsp 页面，可能不只是需要显示某一个 Model 的很多实例列表；还可能需要其他类型 Model 实例显示，可以覆盖方法：

   protected Model setupOneModel(HttpServletRequest request)
3. 自己再优化加工 ModelListForm 方法，覆盖方法：

   public void customizeListForm(ActionMapping actionMapping,
   ActionForm actionForm,
   HttpServletRequest request,
   ModelListForm modelListForm) throws Exception

通过以上方法覆盖实现，基本使用 ModelListAction 可以实现批量查询的各种复杂功能实现，如果你觉得还不行，那么干脆自己模仿 ModelListAction 自己实现一个标准的 struts 的 Action 子类，只要 ActionForm 还是 ModelListForm，页面显示还是可以使用 Jdon 框架的页面标签库。

编写 ModelListAction 子类

com.jdon.strutsutil.ModelListAction.ModelListAction 有两个方法需要实现：

第一．继承 getPageIterator 方法

```
public PageIterator getPageIterator(HttpServletRequest request, int start, int count)
```

这是从服务层获得满足查询条件的当前页面所有 Model 的 ID（主键）集合，当前页面是通过 start 和 count 两个变量说明，前者表示从第几个开始；后者表示开始后需要显示多少个 Model。

以查询商品为例子，ProductListAction 继承 ModelListAction，详细方法内容如下：

```
public class ProductListAction extends ModelListAction {
    public PageIterator getPageIterator(HttpServletRequest request, int start,
```

```
                int count) {
            //获得一个服务
            ProductManager productManager = (ProductManager) WebAppUtil.getService(
                    "productManager", request);
            //从查询参数中获得 categoryId
            String categoryId = request.getParameter("categoryId");
            //调用服务的方法
            return productManager.getProductListByCategory(categoryId);
        }

        public Model findModelByKey(HttpServletRequest request, Object key) {
        ……
        }
    }
```

所以，关键在于服务层的服务需要有个返回参数类型是方法，这是对服务层服务方法的约束要求，这个服务方法如何实现在后面章节介绍。

第二. 继承 findModelByKey 方法

这个方法主要也是从服务层获得一个 Model 实例，ProductListAction 的该方法实现如下：

```
        public Model findModelByKey(HttpServletRequest request, Object key) {
            //获得一个服务
            ProductManager productManager = (ProductManager) WebAppUtil.getService(
                    "productManager", request);
            return productManager. getProduct ((String)key);
        }
```

非常需要注意的是：需要搞清楚这里获得 Model 实例是哪个?

是商品类别 Category 还是商品 Product?

这主要取决于你的批量查询显示的是什么，如果希望显示的是商品 Product 列表，当前这个案例是需要查询某个商品类别 Category 下的所有商品 Product 列表，那么这里的 Model 类型就是 Product。

搞清楚 Model 类型是 Product 后，那么我们可能对 findModelByKey 方法参数 Object 类型的 key 有些疑问，其实这个 key 就是 com.jdon.controller.model. PageIterator 中重要属性 keys（满足查询条件的 ID 集合）中的一个元素，这里根据 key 查询获得一个完整 Model，所以你知道，PageIterator 中封装的不是普通 ID，而是以后需要根据这些 ID 能够获得唯一一个 Model，所以 key 其实是 Model 主键，也是数据表的主键。

以上述案例为例：

productManager. getProduct ((String)key);

这是将 key 类型 Object 强制转换为 String 类型，因为 PageIterator 中 keys（ID 集合）都是 String 类型。所以， key 的类型其实是由你后台创建 PageIterator 时决定的。

第三其他方法：

以上面例子为例，商品批量查询除了显示多个商品 Product 信息外，还需要显示商品目录 Category，也就是说在一个页面中，不但显示多个商品，也要显示商品目录的名称，比如商品目录名是“电器”，其下具体商品列表很多。

在一个页面中装入一个以上 Model 有两种方法：

1. 配置另外一个需要装入 Model 的 ActionForm，例如 Category 对应的 ActionForm 是 CategoryForm，只要在 struts-config.xml 中配置 CategoryForm 的 scope 为 session，同时注意 CategoryForm 在操作流程中要预先赋值。

2. 由于批量查询的 ActionForm 是 ModelListForm，我们可以向这个 ModelListForm 加入一个 Model，实现 ModelListAction 的 customizeListForm 方法, 在这个方法中，实现将 Category 加入的语法：：

```
public void customizeListForm(ActionMapping actionMapping,
                                        ActionForm actionForm,
                                    HttpServletRequest request,
                                   ModelListForm modelListForm ) throws Exception{
        ModelListForm listForm = (ModelListForm) actionForm;
        ProductManager productManager = (ProductManager) WebAppUtil.getService(
                "productManager", request);
        String categoryId = request.getParameter("categoryId");
        Category category = productManager.getCategory(categoryId);
        listForm.setOneModel(category);
}
```

如果加入的布置一个 Model，那么可以使用 DynamicModel 将那些 Model 装载进来，然后在 Jsp 页面再一个个取出来。

struts-config.xml 配置

批量查询与 jdonframework.xml 配置无关，也就是说 jdonframework.xml 中没有与批量查询实现相关的配置，简化了工作，这点与 crud 实现相反，crud 更多的是 jdonframework.xml 配置（当然都少不了 struts-config.xml 配置），crud 缺省情况下除了 Model 和 ModelForm 以外可以没有表现层编码，而批量查询主要是表现层编码，由于查询条件多种多样，只有靠更多编码才能实现查询的灵活性。

上例子中，商品查询的 ModelListForm 配置如同一般 ActionForm 一样配置：

```
  <form-beans>
    <form-bean name="productListForm" type="com.jdon.strutsutil.ModelListForm"/>
    ……
   </form-beans>
```

ProductListAction 配置如同一般 Action 一样：

```
<action-mappings>
    <action path="/shop/viewCategory"
        type="com.jdon.framework.samples.jpetstore.presentation.action.ProductListAction"
        name="productListForm" scope="request"
        validate="false" >
        <forward name="success" path="/catalog/Category.jsp"/>
     </action>
     ……
</action-mappings>
```

Jsp MultiPages 标签

剩余最后工作就是 Jsp 标签使用，通过使用 struts 的 logic:iterator 将前面配置的 productListForm 中的 Model 集合遍历出来。遍历标签语法如下：

```
<logic:iterate indexId="i"     id="user" name="listForm" property="list" >
        <bean:write name="user" property="userId" />
        <bean:write name="user" property="name" />
```

```
</logic:iterate>
```

上述语法将逐个输出每行记录，如果配合 Html 的 table 语法，输出效果如下：

[前页] 1 2 3 4 5 [后页]

主题	发件人	发件单位	日期
just a notice	Sunnny Peng	your team	2003-11-23
just a notice	Sunnny Peng	your team	2003-11-23
just a notice	Sunnny Peng	your team	2003-11-23

批量查询输出还有一个重要功能：页面显示，如上图的“前页”和“后页”显示，这是使用 Jdon 框架的标签库实现的。

只要在 Jsp 页面中你希望显示“前页”和“后页”部位贴入下面代码即可：

```
<%@ taglib uri="/WEB-INF/MultiPages.tld" prefix="MultiPages" %>


…..
<MultiPages:pager actionFormName="listForm" page="/userListAction.do">
<MultiPages:prev>Prev</MultiPages:prev>
<MultiPages:index />
<MultiPages:next>Next</MultiPages:next>
</MultiPages:pager>
```

这是一个 MultiPages 标签，注意标签使用方法前提：

1. 在 Jsp 页面头部需要声明该标签：

```
<%@ taglib uri="/WEB-INF/MultiPages.tld" prefix="MultiPages" %>
```

2. 在 web.xml 中有声明解释该标签库文件所在位置：

```
<taglib>
    <taglib-uri>/WEB-INF/MultiPages.tld</taglib-uri>
    <taglib-location>/WEB-INF/MultiPages.tld</taglib-location>
  </taglib>
```

这表示 Jsp 中的 uri 定义/WEB-INF/MultiPages.tld 实际是执行本地文件/WEB-INF/MultiPages.tld，那么在你的 Web 项目的 WEB-INF 下必须要有 MultiPages.tld 文件，可以从 Jdon 框架源码包目录 src\META-INF\tlds 下将 MultiPages.tld 拷贝过去即可，有些开发工具如 JBuilder 在配置了框架功能后，会自动在建立 Web 项目时拷贝这些标签库的。

下面说明一下批量查询的 MultiPages 标签使用方法：

```
<MultiPages:pager actionFormName="listForm" page="/userListAction.do">
```

其中 actionFormName 是你在 struts-config.xml 中配置的 ModelListForm 名称，也就是这段配置中的 name 值：

```
<form-bean name="listForm" type="com.jdon.strutsutil.ModelListForm"/>
```

MultiPages 中的 page 是指当前页面是通过哪个 action 调用产生的，这样，在显示多页其它页面也可以调用这个 action，这里是/userListAction.do，很显然，这个/userListAction.do 是你的 struts-config.xml 中的 action 配置中的 path 值，如下：

```
<action name="listForm" path="/userListAction" type="com.jdon.framework.test.web.UserListAction"
        scope="request">
        <forward name="success" path="/userList.jsp" />
 </action>
```

上述配置中的 type 值 com.jdon.framework.test.web.UserListAction 是你的 ModelListAction 实现子类。如果你的查询是条件的，也就是说/userListAction.do 后面有查询参数，那么你也需要写在 MultiPages 后面，如：

```
<MultiPages:pager actionFormName="listForm" page="/userListAction.do"
```

```
                paramId="catId" paramName="catId">
```

MultiPages 标签的其它配置很简单，无需变化，如下：

```
<MultiPages:prev>Prev</MultiPages:prev>
<MultiPages:index />
<MultiPages:next>Next</MultiPages:next>
```

MultiPages:prev 是显示“前页”的标签，Prev 是显示“Prev”，你可以使用图标或汉字来替代 Prev。

<MultiPages:index/>是自动显示 1 2 3 …多个数字。目前只提供数字显示。

MultiPages:next 和 MultiPages:prev 类似，显示“后页”字样。

MultiPages:prev 和 MultiPages:next 还有另外一种写法，如下：

```
<MultiPages:prev name="[Prev ]" />
```

使用了 name 属性，这样个语法的好处是：当前页面如果没有超过一页，将没有”Prev”或”Next”字样出现，只有超过一个页面时，才会有相应的字样出现，显示智能化。

开发一个简单 Jdon 应用系统

开发环境：JBuilder /eclipse+ JBoss（演示版本是 JBuilder 2005 和 JBoss 3.2X）。

以下以一个简单需求说明 Struts+Jdon+EJB/POJO 的开发，源码见 Jdon 框架源码包中的 SimpleJdonFrameworkTest/JdonFrameworkTest 项目，更复杂一些的案例可见源码包中的另外一个 JdonNews，简单需求如下：

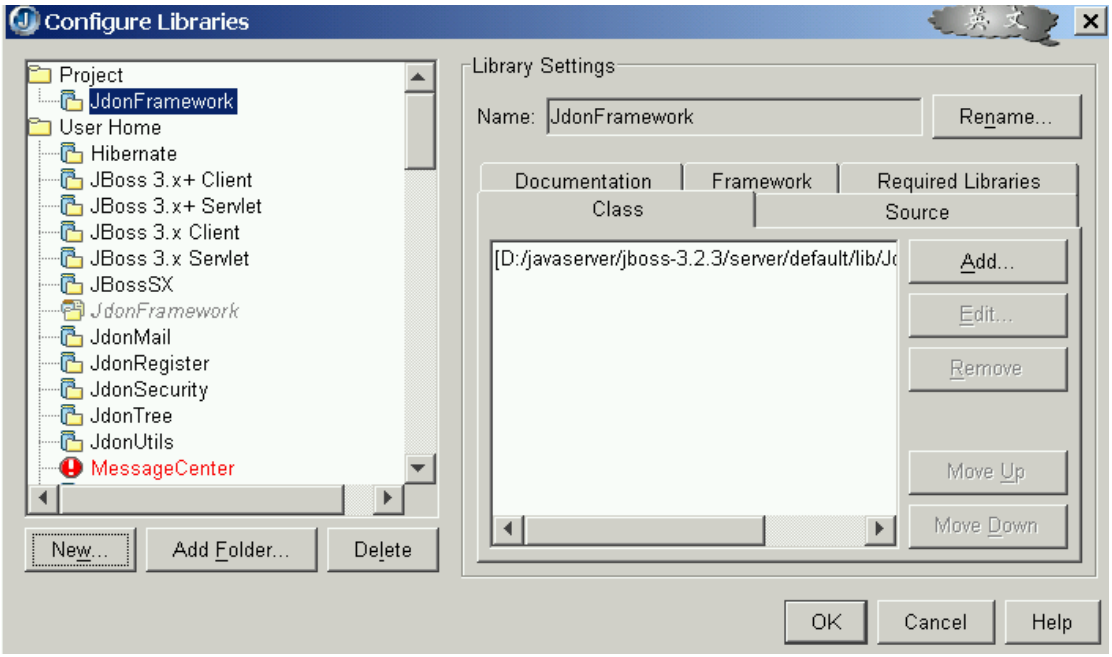
实现 User 模型的数据新增、修改、删除和批量查询。

User 模型有两个字段 userId 和 name。

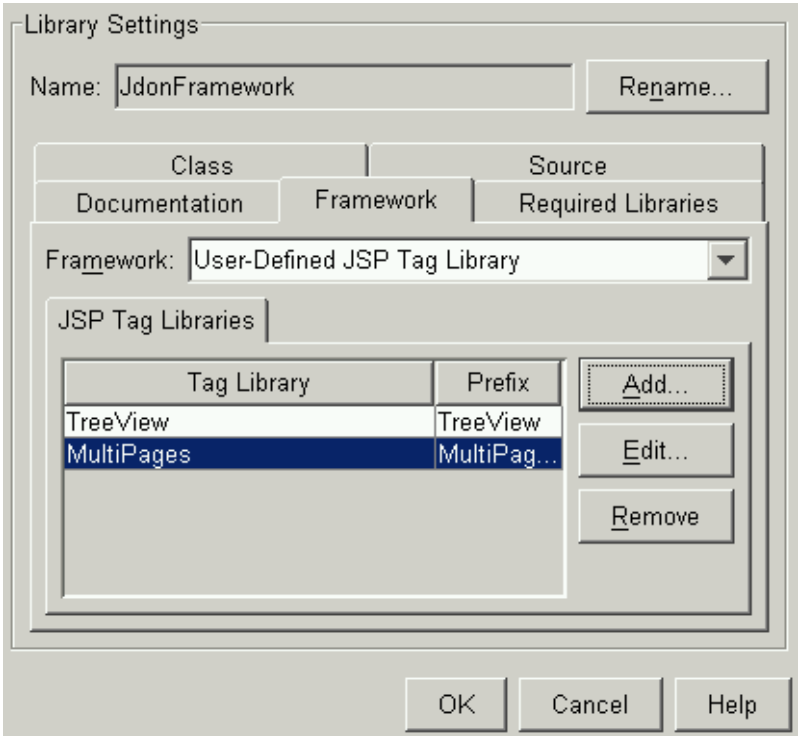
在 JBuilder 配置 JdonFramework 为 IDE 驱动包

如果你使用 JBuilder 开发工具，按下列步骤：

将 JdonFramework 导入项目库，或者成为整个配置库。如图：



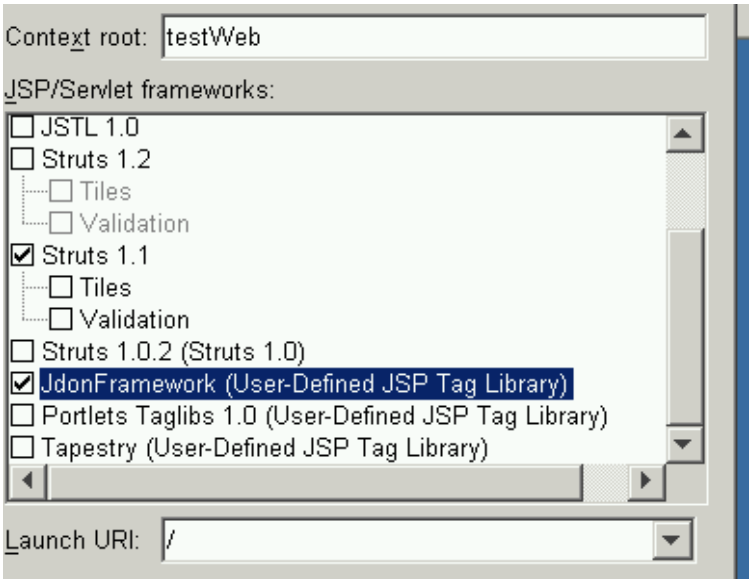
同时，配置 Framework 配置，如下图：



该配置是为了激活项目 Web 开发支持 JdonFramework 的标签库 taglib，使用 JdonFramework 开发应用系统时，自动支持多页批量查询和树形结构显示，这两种功能需要在 Jsp 页面导入特定的标签库，该设置就是为实现此功能。

注意：由于 JBuilder 的部分 BUG，最好将 JdonFramework.jar 的 MEAT-INF/tlds 目录下的 MultiPages.tld 等文件解压出，形成单独文件，在这里导入这些单独文件。

配置 WEB 模块时，需选中 Web 框架，如下图：



如果你使用的是 Eclipse，在将 JdomFramework.jar 作为一般的 JAR 导入项目。

建模

开始开发程序， 建立 Model 实现 UserTest

建立 Model 对应的 ModelForm：UserActionForm 是继承 ModelForm，而 ModelForm 实际也是继承 Struts 的 ActionForm。

将 UserTest 中方法和字段拷贝到 UserActionForm，原则上这两者内容是一致的，JdonFramework 将自动在这两个对象之间拷贝相同方法的值，但是它们服务的对象不一样，UserTest 是为了中间层或后台业务层服务的；而 UserActionForm 是为了界面服务的，两者分离是为了使得我们的业务层和界面分离，这样，当有新的可替代 Struts 界面框架出现，我们可以不修改业务层代码下更换界面框架。

业务层开发

Jdon 框架支持 Web 应用和 EJB 应用，下面两种服务实现任意选一种。

POJO 服务实现

POJO 服务只要编写一个 TestService 接口和一个实现 TestServicePOJOImp 即可，在实现子类中调用持久层 jdbcDao.

EJB 服务实现

这里使用 EJB 实现业务层功能，建立一个无态 Session Beans

为 TestEJB，主要负责 UserTest 的数据库增删改查等功能，这部分开发不在 JdonFramework 规定之类，而要遵循 EJB 相关规定。这部分开发细节可参见有关 JBuilder 开发 EJB 教程。

建立一个实体 Bean 名为 User：首先需要将 JBuilder 配置成能够直接访问数据库，我们使用 MySQL 数据库，配置后，重新启动 JBuilder。

配置 JBoss 中的数据源，数据源 JNDI 名称是 DefaultDS。在代码中调用 JBoss 的 JNDI，JBoss 有特定规定，应该是 java:/DefaultDS。

创建 CMP 时有两种方式：一种直接创建新的 CMP，当该 J2EE 部署时，将自动创建 CMP 对应的数据表，或者先创建数据表，由数据表导入 CMP。本演示采取后者方式。

完成 Session Bean TestEJB 的新增、修改和删除方法，下面完成查询和批量查询方法，查询方法建议使用 DAO+JDBC 实现。

持久层：增删改查 crud 实现

持久层的增删改查功能可使用 Jdon 框架的 Jdbc 模板实现：

新增操作

```
public void insert(UserTest userTest) throws Exception{
    String sql = "INSERT INTO testuser (userId , name) "+
    "VALUES (?, ?)";
    List queryParams = new ArrayList();
    queryParams.add(userTest.getUserId());
    queryParams.add(userTest.getName());
    jdbcTemp.operate(queryParams,sql);
    clearCacheOfItem();
}
```

修改操作：

```
public void update(UserTest userTest) throws Exception{
        String sql = "update testuser set name=? where userId=?";
        List queryParams = new ArrayList();
        queryParams.add(userTest.getName());
        queryParams.add(userTest.getUserId());
        jdbcTemp.operate(queryParams,sql);
        clearCacheOfItem();
}
```

删除操作：

```
public void delete(UserTest userTest) throws Exception{
        String sql = "delete from testuser where userId=?";
        List queryParams = new ArrayList();
        queryParams.add(userTest.getUserId());
        jdbcTemp.operate(queryParams,sql);
        clearCacheOfItem();
}
```

查询操作：

```
public UserTest getUser(String Id) {
        String GET_FIELD = "select    * from testuser where userId = ?";
        List queryParams = new ArrayList();
        queryParams.add(Id);

        UserTest ret = null;

        try {
            List list = pageIteratorSolverOfUser.queryMultiObject(queryParams,
                    GET_FIELD);
            Iterator iter = list.iterator();
            if (iter.hasNext()) {
                Map map = (Map) iter.next();
                ret = new UserTest();
                ret.setName((String) map.get("name"));
                ret.setUserId((String) map.get("userId"));
            }
        } catch (Exception se) {
        }
        return ret;
  }
```

持久层：批量查询实现

这里是为了实现批量查询，如果不需要批量查询，可跳过此步：

持久层需要向前台提供数据库等持久层数据的查询和获得，一般可通过 Jdon 框架的简单 JDBC 模板实现，我们创建一个 JdbcDao 类。

PageIterator 对象创建

首先为每个 Model 引入分页对象；

```
//通过 JNDI 获得 Datasource
dataSource = (DataSource) sl.getDataSource(JNDINames.DATASOURCE);
//框架使用 为每个 Model 建立一个分页对象。
pageIteratorSolverOfUser = new PageIteratorSolver(dataSource);
```

三个方法实现

JdbcDao 有三个缺省方法必须才能完成从数据库中获得所要求的数据，这三个方法分别是：

```
User getUsers(String Id);       //查询获得单个 Model 实例
PageIterator getUsers(int start, int count); //查询某页面的所有 Model 的 ID 集合
public void clearCacheOfUser() ;  //清除某个 Model 的缓存。
```

上述三个方法中，只有一个方法需要编写代码，其它可委托 JdonFramework 提供的工具 API 方法完成。

第一个方法完成： 编写获得单个 Usertest 的数据库查询方法：从现成的示例拷贝如下：

```
public UserTest getUser(String Id) {
    String GET_FIELD = "select    * from testuser where userId = ?";
    List queryParams = new ArrayList();
    queryParams.add(Id);
    UserTest ret = null;
    try {
        List list = pageIteratorSolverOfUser.queryMultiObject(queryParams,
                GET_FIELD);
        Iterator iter = list.iterator();
        if (iter.hasNext()) {
            Map map = (Map) iter.next();
            ret = new UserTest();
            ret.setName((String) map.get("name"));
            ret.setUserId((String) map.get("userId"));
        }
    } catch (Exception se) {
    }
    return ret;
}
```

第二个方法实现：编写分页查询 SQL 实现方法，直接使用框架内部的方法如下：

```
    String GET_ALL_ITEMS_ALLCOUNT = "select count(1) from testuser ";

    String GET_ALL_ITEMS = "select userId    from testuser ";

    return pageIteratorSolverOfUser.getDatas("", GET_ALL_ITEMS_ALLCOUNT,
            GET_ALL_ITEMS, start, count);
```

注意 getDatas 的方法参数使用，参考批量查询章节

第三个方法实现.：编写缓存清除方法：该方法是在该 Model 被新增或删除等情况下调用。 这样，当前台页面新增新的数据后，批量查询能够立即显示新的数据，否则将是缓存

中老的数据集合，无法立即更新。

```
public void clearCacheOfUser() {
    pageIteratorSolverOfUser.clearCache();
  }
```

JdbcDAO 模板化编程到此结束。

下一步，使用 EJB 的 Session Bean 包装这个 JdbcDao，如果服务层不是使用 EJB 实现，直接通过一般的 POJO 服务包装 JdbcDao。

JdonFramework.xml 配置

根据 http://www.jdon.com/ jdonframework.dtd 导出 XML:

增删改查 crud 配置

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE app PUBLIC "-//JDON//DTD Framework 2005 1.0 //EN"
"http://www.jdon.com/jdonframework.dtd">
<app>
  <models>
    <model key="userId" class ="com.jdon.framework.test.model.UserTest">
      <actionForm name="userActionForm"/>
      <handler>
        <service ref="testService">
          <getMethod name="getUser" />
          <createMethod name="createUser" />
          <updateMethod name="updateUser" />
          <deleteMethod name="deleteUser" />
        </service>
      </handler>
    </model>
  </models>
  ….. <!--  服 务 配置  -->
</app>
```

注意 Handler 配置有两种：缺省是上述配置，如果情景复杂，需要代码完成，则编写一个 Hamdler 类继承 ModelHandler 实现即可，然后在上述 handler 段配置如下：

<handler class=”xxxx.yourHandler” />

POJO 服务配置实现

```
  <services>
    <pojoService name="testService" class="com.jdon.framework.test.service.TestServicePOJOImp"/>
    <pojoService class="com.jdon.framework.test.dao.JdbcDAO" name="jdbcDAO">
      <constructor value="java:/DefaultDS"/>
    </pojoService>
  </services>
```

上面定义了一个 POJO 服务：com.jdon.framework.test.service.TestServicePOJOImp，因为 TestServicePOJOImp 中使用了 JdbcDAO，这里配置了 com.jdon.framework.test.dao.JdbcDAO

实现，如果你使用其他持久层技术，可以在此更换另外一个 JdbcDao。

这里的jdbcDAO需要数据库连接，数据库连接参数不是在应用程序中配置，而是使用J2EE服务器已经配置成功的JNDI名称，Tomcat中配置JNDI数据库连接池可见：http://www.7880.com/Info/Article-37f05fa0.html；JBoss的JND配置见本站相关文章。

java:/DefaultDS 是 JBoss 的数据库连接池 JNDI 写法，如果你使用其他服务器，参考他们的 JNDI 要求写法。

如果服务配置中存在 ejbService，如下：

```
<ejbService name="testService2" >
  <jndi name="TestEJB" />
  <ejbLocalObject class="com.jdon.framework.test.ejb.TestEJBLocal"/>
  <interface class="com.jdon.framework.test.service.TestService" />
</ejbService>
```

那么可以在 ejbService 和 pojoService 之间更换，将 pojoService 和 ejbService 的 name 值对换一下即可， 将<ejbService name="testService2" >更换为<ejbService name="testService" >，将<pojoService name="testService"改名为 <pojoService name="testService2"， 这样，Model 部分配置的服务指向就从调用 POJO 服务更换到调用 EJB 服务了。

表现层：增删改查 crud 配置

crud 功能无需编制代码，代码功能已经在上节的 jdonframework.xml 中配置完成，这里只要配置 struts 的页面流程 struts-config.xml 配置和编写 Jsp 两步即可。

(1) 按照前面章节规则二：配置 ModelViewAction 介绍的。实现新增或编辑视图输出的 Action:

```
<action name="userActionForm" path="/userAction" type="com.jdon.strutsutil.ModelViewAction">
  <forward name="create" path="/user.jsp" />
  <forward name="edit" path="/user.jsp" />
</action>
```

上述配置中，需要修改的跟 Model 名称相关的配置。

（2）创建 user.jsp 页面。

在 user.jsp 中要增加一行关键语句：

<html:hidden property="action"/> 用来标识新增 修改等动作。

（3） 配置 userSaveAction 实现数据真正操作。

```
<action name="userActionForm" path="/userSaveAction"
        type="com.jdon.strutsutil.ModelSaveAction">
  <forward name="success" path="/result.jsp" />
  <forward name="failure" path="/result.jsp" />
</action>
```

（4） 创建数据操作结果页面 result.jsp， 上述配置修改不大，需要修改的跟 Model 名称相关的配置。

表现层：批量查询实现

crud 功能只需配置和 Jsp 实现就可以，而批量查询表现层需要进简单编码，然后再需要

配置和 Jsp 完成。还有一个区别是：前者无需编码，替代的是需要配置 jdonframework.xml；而后者因为编码实现；无需配置 jdonframework.xml 了。

编码：UserListAction

UserListAction 继承 ModelListAction 实现，按照前面批量查询章节，这里完成两个方法：getPageIterator 和 findModelByKey：

getPageIterator 方法如下：

```
public PageIterator getPageIterator(HttpServletRequest request, int start,
                                    int count) {
    PageIterator pageIterator = null;
    try {
        TestService testService = (TestService) WebAppUtil.getService("testService",  request);
       pageIterator = testService.getAllUsers(start, count);
    } catch (Exception ex) {
        logger.error(ex);
    }
    return pageIterator;
}
```

findModelByKey 方法：

```
public Model findModelByKey(HttpServletRequest request, Object key) {
    Model model = null;
    try {
        TestService testService = (TestService) WebAppUtil.getService("testService",  request);
       model = testService.getUser( (String) key);
    } catch (Exception ex) {
        logger.error(ex);
    }
    return model;


}
```

代码中：

```
TestService testService = (TestService) WebAppUtil.getService("testService",  request);
```

其中 testService 是 jdonframework.xml 配置中<pojoService name=" testService "> 中名称。

Struts-config.xml 配置

（1）配置一个批量查询的 ActionForm： 首先在 struts-config.xml 创建一个用于批量分页查询的 ActionForm，固定推荐名称为 listForm。

```
<form-bean name="listForm" type="com.jdon.strutsutil.ModelListForm" />
```

（2）配置 UserListAction，按照普通 Struts 的 action 配置

```
<action name="listForm" path="/userListAction" type="jdonframeworktest.web.UserListAction" >
   <forward name="success" path="/userList.jsp" />
</action>
```

（4）编写 userList.jsp，用于分页显示查询结果。

在 userList.jsp 中使用到分页的标签库：

```
<%@ taglib uri="/WEB-INF/MultiPages.tld" prefix="MultiPages" %>
```

```
<MultiPages:pager actionFormName=" Struts-config.xml 中 formBeans 名称：listForm"
page="Struts-config.xml 中当前的 action path 值：/userListAction.do">
<MultiPages:prev>前页</MultiPages:prev>
<MultiPages:index />
<MultiPages:next>后页</MultiPages:next>
</MultiPages:pager>
```

我们将 userList.jsp 作为本演示首页，将新增 修改和删除等操作的执行集中在这个页面实现，使用 JavaScript 来实现。

基本全部完成。

配置启动 Jdon 框架

（1）配置 jdonframework.xml 在 struts-config.xml 中。

```
<plug-in className="com.jdon.strutsutil.InitPlugIn">
  <set-property property="modelmapping-config"  value="jdonframework.xml 完整路径" />
</plug-in>
```

（2）将 jdonframework.xml 打包到你的 Web 项目。

否则，后台控制台会出现：

[InitPlugIn] looking up a config: jdonframework.xml

[InitPlugIn] cannot locate the config:: jdonframework.xml

打包部署

通过 JBuilder 的 new 创建，建立一个 ear 应用系统。

注意，将 Web 项目中去除任何包，也就是说 WEB-INF/lib 下没有任何包，Struts 和 JdonFramework 包都放置在 JBoss 的 server/default/lib 下。

如果出现与 tld 相关的错误， 如：

File "/WEB-INF/MultiPages.tld" not found

解决：在 web.xml 中应该有下面两行 tld 配置：

```
<taglib>
  <taglib-uri>/WEB-INF/MultiPages.tld</taglib-uri>
  <taglib-location>/WEB-INF/MultiPages.tld</taglib-location>
</taglib>
<taglib>
  <taglib-uri>/WEB-INF/TreeView.tld</taglib-uri>
  <taglib-location>/WEB-INF/TreeView.tld</taglib-location>
</taglib>
```

并且确保/WEB-INF/MultiPages.tld 和/WEB-INF/TreeView.tld 下文件存在，这两个 MultiPages.tld 和 TreeView.tl 在 JdonFramework.jar 中的 META-INF/tlds 中，可手工拷贝过去。

更多详细错误，打开 JBoss 日志文件：server/default/log/server.log

运行案例

打开浏览器，键入http://localhost:8080/testWeb/index.jsp

我们将 index.jsp 导向真正执行/userListAction.do

网上实时演示网址： http://www.jdon.com:8080/testWeb/

本章节源码在 Jdon 框架源码包的 samples 中 SimpleJdonFrameworkTest/JdonFrameworkTest

J2EE 应用系统 Jdon JPetstore

以 IBatis.com 的 iBATIS-Jpetstore 为例，我们使用 Jdon 框架对其重构成为 Jdon-JPetstore，本章开发环境是 Eclipse（本章案也适用其他开发工具），部署运行环境是 JBoss。

Eclipse 是一个非常不错的开源开发工具，使用 Eclipse 开发和使用 JBuilder 将有完全不同的开发方式。我们使用 Eclipse 基于 Jdon 框架开发一个完全 Web 应用，或者可以说，开发一个轻量（lightweight）的 J2EE 应用系统。

通过这个轻量系统开发，说明 Jdon 框架对完全 POJO 架构的支持，因为 EJB 分布式集群计算能力，随着访问量提升，可能需要引入 EJB 架构，这时只要使用 EJB session Bean 包装 POJO 服务则可以无缝升级到 EJB。使用 Jdon 框架可实现方便简单地架构升迁。

Eclipse 安装简要说明

1.下载 Eclipse:在 http://www.eclipse.org 的下载点中选择 tds ISP 比较快。

2.安装免费插件：

编辑 Jsp 需要 lomboz ： http://www.objectlearn.com/projects/download.jsp

注意对应的 Eclipse 版本。

编辑 XML，使用 Xmlbuddy: http://xmlbuddy.com/

基本上这两个插件就够了。

如果希望开发 Hibernate，插件：

http://www.binamics.com/hibernatesync

代码折叠

http://www.coffee-bytes.com/eclipse/update-site/site.xml

3. 关键学习 ant 的编写 build.xml，在 build.xml 将你的 jsp javaclass 打包成 war 或 jar 或 ear 就可以。都可以使用 ant 的 jar 打包，build.xml 只要参考一个模板就可以：SimpleJdonFrameworkTest.rar 有一个现成的，可拷贝到其它项目后修改后就可用.

然后在这个模板上修改，参考 ant 的命令参考:

http://ant.apache.org/manual/tasksoverview.html

网上有中文版的 ant 参考，在 google 搜索就能找到。

关键是学习 ant 的 build.xml 编辑，SimpleJdonFrameworkTest.rar 有一个现成的，可拷贝到其它项目后修改后就可用.

用 ant 编译替代 Eclipse 的缺省编译：选择项目属性-->Builders ---> new --> Ant Builder --->选择本项目的 build.xml workspace 选择本项目

eclipse 开发就这些，非常简单，不象 Jbuilder 那样智能化，导致项目目录很大。eclipse 只负责源码开发，其它都由 ant 负责

架构设计要点

Jdon-JPetstore 除了保留 iBATIS-JPetstore 4.0.5 的域模型、持久层 ibatis 实现以及 Jsp 页面外，其余部分因为使用了 Jdon 框架而和其有所不同。

保留域模型和 Jsp 页面主要是在不更改系统需求的前提下，重构其架构实现为 Jdon 框架，通过对比其原来的实现或 Spring 的 JPetstore 实现，可以发现 Jdon 框架的使用特点。

在原来 jpetstore iBatis 包会延伸到表现层，例如它的分页查询 PaginatedList，iBatis 只是持久层框架，它的作用范围应该只限定在持久层，这是它的专业范围，如果超过范围，显得 ….。所以，在 Jdon-Jpetstore 中将 iBatis 封装在持久层（砍掉 PaginatedList 这只太长的手），Jdon 框架是一种中间层框架，联系前后台的工作应该由 Jdon 这样的中间层框架完成。

在iBatis 4.0.5 版本中，它使用了一个利用方法映射Reflection的小框架，这样，将原来需要在Action实现方法整入了ActionForm中实现，ActionForm变成了一个复杂的对象：页面表单抽象以及与后台Service交互，在ActionForm中调用后台服务是通过单态模式实现，这是在一般 J2EE 开发中忌讳的一点，关于单态模式的讨论可见：http://www.jdon.com/jive/article.jsp?forum=91&thread=17578

Jdon 框架使用了微容器替代单态，消除了 Jpetstore 的单态隐患，而且也简化了 ActionForm 和服务层的交互动作（通过配置实现）。

用户注册登陆模块实现

用户域建模（Model）

首先，我们需要从域建模开始，建立正确的领域模型，以用户账号为例，根据业务需求我们确立用户账号的域模型 Account，该模型需要继承 Jdon 框架中的 com.jdon.controller.model.Model。

```
public class Account extends Model {

   private String username;
   private String password;
   private String email;
   private String firstName;
   private String lastName;
   ……

}
```

username 是主键。

域模型建立好之后，就可以花开两朵各表一支，表现层和持久层可以同时开发，先谈谈持久层关于用户模型的 crud 功能实现。

持久层 Account crud 实现

主要是用户的新增和修改，主要用于注册新用户和用户资料修改。

```
public interface AccountDao {
   Account getAccount(String username);   //获得一个 Account
```

```
    void insertAccount(Account account);    //新增
    void updateAccount(Account account); //修改
}
```

持久层可以使用多种技术实现，例如 Jdon 框架的 JdbcTemp 代码实现比较方便，如果你的 sql 语句可能经常改动，使用 iBatis 的 sql 语句 XML 定义有一定好处，本例程使用 Jpetstore 原来的持久层实现 iBatis。见源码包中的 Account.xml

表现层 Account 表单创建（ModelForm）

这是在 Domain Model 建立后最重要的一步，是前台表现层 Struts 开发的起步，表单创建有以下注意点：

表单类必须继承 com.jdon.model.ModelForm

表单类基本是 Domain Model 的影子，每一个 Model 对应一个 ModelForm 实例，所谓对应：就是字段名称一致。ModelForm 实例是由 Model 实例复制获得的。

```
public class AccountForm extends ModelForm {

    private String username;
    private String password;
    private String email;
    private String firstName;
    private String lastName;
    ……

}
```

当然 AccountForm 可能有一些与显示有关的字段，例如注册时有英文和中文选择，以及类别的选择，那么增加两个字段在 AccountForm 中：

```
    private List languages;
    private List categories;
```

这两个字段需要初始化值的，因为在 AccountForm 对应的 Jsp 的页面中要显示出来，这样用户才可能进行选择。选择后的值将放置在专门的字段中。

有两种方式初始化这两个字段：

1. 在 AccountForm 构造方法中初始化，前提是：这些初始化值是常量，如：

```
public AccountForm() {
    languages ＝ new ArrayList();
    languages.add("english");
    languages .add("japanese");
}
```

2.如果初始化值是必须从数据库中获取，那么采取前面章节介绍的使用 ModelHandler 来实现，这部分又涉及配置和代码实现，缺省时我们考虑通过 jdonframework.xml 配置实现。

Account crud 的 struts-config.xml 的配置

第一步配置 ActionForm：

上节编写了 ModelForm 代码，ModelForm 也就是 struts 的 ActionForm，在 struts-config.xml

中配置 ActionForm 如下：

```
<form-bean name="accountFrom"
            type="com.jdon.framework.samples.jpetstore.presentation.form.AccountForm"/>
```

第二步配置 Action：

这需要根据你的 crud 功能实现需求配置，例如本例中用户注册和用户修改分开，这样，配置两套 ModelViewAction 和 ModelSaveAction，具体配置见源码包中的 struts-config-security.xml，这里将 struts-config.xml 根据模块划分成相应的模块配置，实现多模块开发，本模块是用户注册登陆系统，因此取名 struts-config-security.xml。

Account crud 的 Jdonframework.xml 配置

```
<model key="username" class ="com.jdon.framework.samples.jpetstore.domain.Account">
  <actionForm name="accountForm"/>
  <handler>
    <service ref="accountService">
      <initMethod     name="initAccount" />
      <getMethod      name="getAccount" />
      <createMethod name="insertAccount" />
      <updateMethod name="updateAccount" />
      <deleteMethod name="deleteAccount" />
    </service>
  </handler>
</model>
```

.其中有一个 initMethod 主要用于 AccuntForm 对象的初始化。其他都是增删改查的常规实现。

Account crud 的 Jsp 页面实现

在编辑页面 EditAccountForm.jsp 中加入:

<html:hidden name="accountFrom" property="action" value="create" />

在新增页面 NewAccountForm.jsp 加入：

<html:hidden name="accountFrom" property="action" value="edit" />

所有的字段都是直接来自 accountFrom。

整理模块配置

商品模块功能完成，struts 提供了多模块开发，因此我们可以将这一模块单独保存在一个配置中：/WEB-INF/struts-config-security.xml，这样以后扩展修改起来方便。

商品查询模块实现

在 iBATIS-JPetstore 中没有单独的 CategoryForm，而是将三个 Model：Category、Product、，Item 合并在一个 CatalogBean 中，这样做的缺点是拓展性不强，将来这三个 Model 也许需要单独的 ActionForm。

由于我们使用 Jdon 框架的 crud 功能配置实现，因此，不怕细分这三个 Model 带来代码

复杂和琐碎。

由于原来的 Jpetstore“偷懒”，没有实现 Category Product 等的 crud 功能，只实现它们的查询功能，因此，我们使用 Jdon 框架的批量查询来实现查询。

持久层 Product 批量查询实现

商品查询主要有两种批量查询，根据其类别 ID：CategoryId 查询所有该商品目录下所有的商品；根据关键字搜索符合条件的所有商品，下面以前一个功能为例子：

iBatis-jpetstore 使用 PaginatedList 作为分页的主要对象，该对象需要保存到 HttpSession 中，然后使用 PaginatedList 的 NextPage 等直接遍历，这种方法只适合在小数据量合适，J2EE 编程中不推荐向 HttpSession 放入大量数据，不利于 cluster。

根据 Jdon 批量查询的持久层要求，批量查询需要两种 SQL 语句实现：符合条件的 ID 集合和符合条件的总数：以及单个 Model 查询。

```
//获得 ID 集合
List getProductIDsListByCategory(String categoryId, int pagessize);
//获得总数
int getProductIDsListByCategoryCount(String categoryId);
//单个 Model 查询
Product getProduct(String productId) ;
```

这里我们需要更改一下 iBatis 原来的 Product.xml 配置，原来，它设计返回的是符合条件的所有 Product 集合，而我们要求是 Product ID 集合。

修改 Product.xml 如下：

```
<resultMap id="productIDsResult" class="java.lang.String">
    <result property="value" column="PRODUCTID"/>
</resultMap>

<select id="getProductListByCategory" resultMap="productIDsResult" parameterClass="string">
    select PRODUCTID from PRODUCT where CATEGORY = #value#
</select>

<select id="getProductListByCategoryCount"   resultClass="java.lang.Integer" parameterClass="string">
    select count(1) as value from PRODUCT where CATEGORY = #value#
</select>
```

ProductDao 是 IBatis DAO 实现，读取 Product.xml 中配置：

```
public List   getProductIDsListByCategory(String categoryId, int start, int pagessize) {
    return sqlMapDaoTemplate.queryForList(
            "getProductListByCategory", categoryId, start, pagessize);
}

public int getProductIDsListByCategoryCount(String categoryId){
    Integer countI = (Integer)sqlMapDaoTemplate.queryForObject(
            "getProductListByCategoryCount", categoryId);
    return countI.intValue();
}
```

这样，结合配置的 iBatis DAO 和 Jdon 框架批量查询，在 ProductManagerImp 中创建 PageIterator，当然这部分代码也可以在 ProductDao 实现，创建 PageIterator 代码如下：

```
public PageIterator getProductIDsListByCategory(String categoryId, int start, int count)
{
```

```
        PageIterator pageIterator = null;
         try {
               List list = productDao.getProductIDsListByCategory(categoryId, start, count);
               int allCount = productDao.getProductIDsListByCategoryCount(categoryId);
               int currentCount = start + list.size();
               pageIterator = new PageIterator(allCount, list.toArray(), start,
                        (currentCount < allCount)?true:false);

         } catch (DaoException daoe) {
              Debug.logError(" Dao error : " + daoe, module);
         }

       return pageIterator;
```

表现层 Product 批量查询实现

根据批量查询的编程步骤，在表现层主要是实现 ModelListAction 继承、配置和 Jsp 编写，下面分步说：

第一步，创建一个 ModelListAction 子类 ProductListAction，实现两个方法：getPageIterator 和 findModelByKey，getPageIterator 方法如下：

```
    public PageIterator getPageIterator(HttpServletRequest request, int start,
            int count) {
        ProductManager productManager = (ProductManager) WebAppUtil.getService(
                "productManager", request);
        String categoryId = request.getParameter("categoryId");
        return productManager.getProductIDsListByCategory(categoryId, start, count);

    }
```

findModelByKey 方法如下：

```
    public Model findModelByKey(HttpServletRequest request, Object key) {
        ProductManager productManager = (ProductManager) WebAppUtil.getService(
                "productManager", request);
        return productManager.getProduct((String)key);
    }
```

由于我们实现的是查询一个商品目录下所有商品功能，因此，需要显示商品目录名称，而前面操作的都是 Product 模型，所以在显示页面也要加入商品目录 Category 模型，我们使用 ModelListAction 的 customizeListForm 方法：

```
    public void customizeListForm(ActionMapping actionMapping,
            ActionForm actionForm, HttpServletRequest request,
            ModelListForm modelListForm) throws Exception {
        ModelListForm listForm = (ModelListForm) actionForm;
        ProductManager productManager = (ProductManager) WebAppUtil.getService(
                "productManager", request);
        String categoryId = request.getParameter("categoryId");
        Category category = productManager.getCategory(categoryId);
        listForm.setOneModel(category);
    }
```

第二步，配置 struts-config.xml，配置 ActionForm 和 Action：

```
    <form-bean name="productListForm" type="com.jdon.strutsutil.ModelListForm"/>
```

action 配置如下：

```
<action path="/shop/viewCategory"
        type="com.jdon.framework.samples.jpetstore.presentation.action.ProductListAction"
  name="productListForm" scope="request"
  validate="false" >
  <forward name="success" path="/catalog/Category.jsp"/>
 </action>
```

第三步，编写 Category.jsp

从 productListForm 中取出我们要显示两个模型，一个是 oneModel 中的 Category；另外一个是 Product Model 集合 list，Jsp 语法如下：

```
<bean:define id="category" name="productListForm" property="oneModel" />
<bean:define id="productList" name="productListForm" property="list" />
```

我们可以显示商品目录名称如下：

```
<h2><bean:write name="category" property="name" /></h2>
```

这样我们就可以遍历 productList 中的 Product 如下：

```
<logic:iterate id="product" name="productList" >
  <tr bgcolor="#FFFF88">
  <td><b><html:link paramId="productId" paramName="product" paramProperty="productId"
page="/shop/viewProduct.shtml"><font color="BLACK"><bean:write name="product" property="productId"
/></font></html:link></b></td>
  <td><bean:write name="product" property="name" /></td>
  </tr>
</logic:iterate>
```

加上分页标签库如下：

```
  <MultiPages:pager actionFormName="productListForm " page="/shop/viewCategory.do"
            paramId="categoryId" paramName="category" paramProperty="categoryId">
<MultiPages:prev><img src="../images/button_prev.gif" border="0"></MultiPages:prev>
<MultiPages:index />
<MultiPages:next><img src="../images/button_next.gif" border="0"></MultiPages:next>
</MultiPages:pager>
```

至此，一个商品目录下的所有商品批量查询功能完成，由于是基于框架的模板化编程，直接上线运行成功率高。

商品搜索批量查询:

参考上面步骤，商品搜索也可以顺利实现，从后台到前台按照批量查询这条线索分别涉及的类有：

持久层实现：ProductDao 中的三个方法：

```
List searchProductIDsList(String keywords, int start, int pagessize); //ID 集合

int searchProductIDsListCount(String keywords); //总数

Product getProduct(String productId) ; //单个 Model
```

表现层：建立 ProductSearchAction 类，配置 struts-config.xml 如下：

```
<action path="/shop/searchProducts"
        type="com.jdon.framework.samples.jpetstore.presentation.action.ProductSearchAction"
  name="productListForm" scope="request"
  validate="false">
```

```
            <forward name="success" path="/catalog/SearchProducts.jsp"/>
        </action>
```

与前面使用的都是同一个 ActionForm：productListForm

编写 SearchProducts .jsp，与 Category.jsp 类似，相同的是 ActionForm；不同的是 action。

商品条目 Item 批量查询

条目 Item 批量实现与 Product 批量查询类似：

持久层：ItemDao 提供三个方法：

```
List getItemIDsListByProduct(String productId, int start, int pagessize);//ID 集合

  int getItemIDsListByProductCount(String productId);//总数

  Item getItem(String itemId); //单个 Model
```

表现层：创建一个 ItemListAction 继承 ModelListAction: 完成 getPageIterator 和 findModelByKey，如下：

```
public class ItemListAction extends ModelListAction {

    public PageIterator getPageIterator(HttpServletRequest request, int start,
            int count) {
        ProductManager productManager = (ProductManager) WebAppUtil.getService(
                "productManager", request);
        String productId = request.getParameter("productId");
        return productManager.getItemIDsListByProduct(productId, start, count);
    }

    public Model findModelByKey(HttpServletRequest request, Object key) {
        ProductManager productManager = (ProductManager) WebAppUtil.getService(
                "productManager", request);
        return productManager.getItem((String)key);
    }

    public void customizeListForm…….
}
```

与前面的 ProductListAction 相比，非常类似，不同的是 Model 名称不一样，一个是 Product 一个是 Item;

struts-config.xml 配置如下：

```
    <form-bean name="itemListForm" type="com.jdon.strutsutil.ModelListForm"/>

    <action path="/shop/viewProduct"
            type="com.jdon.framework.samples.jpetstore.presentation.action.ItemListAction"
      name="itemtListForm" scope="request"
      validate="false">
      <forward name="success" path="/catalog/Product.jsp"/>
    </action>
```

比较前面 product 的配置，非常类似，其实 itemListForm 和 productListForm 是同一个 ModelListForm 类型，可以合并起来统一命名为 listForm，节省 ActionForm 的配置。

Product.jsp 页面与前面的 Category.jsp SearchProdcuts.jsp 类似。

```
<bean:define id="product" name="itemListForm" property="oneModel" />
<bean:define id="itemList" name="itemListForm" property="list" />
```

分页显示：

```
<MultiPages:pager actionFormName="itemListForm" page="/shop/viewProduct.do"
                  paramId="productId" paramName="product" paramProperty="productId">
  …..
```

商品条目 Item 单条查询

单个显示属于 crud 中的一个查询功能，我们需要建立 Model 对应的 ModelForm，将 Item 的字段拷贝到 ItemForm 中。配置这个 ActionForm 如下：

```
<form-bean name="itemForm"
           type="com.jdon.framework.samples.jpetstore.presentation.form.ItemForm"/>
```

第二步：因为这个功能属于 crud 一种，无需编程，但是需要配置 jdonframework.xml：

```
<model key="itemId" class ="com.jdon.framework.samples.jpetstore.domain.Item">
  <actionForm name="itemForm"/>
  <handler>
    <service ref="productManager">
      <getMethod name="getItem" />
    </service>
  </handler>
</model>
```

配置中只要一个方法 getMethod 就可以，因为只用到 crud 中的读取方式。

第三步：配置 struts-config.xml 如下：

```
<action    path="/shop/viewItem"    type="com.jdon.strutsutil.ModelDispAction"
 name="itemForm" scope="request"
 validate="false">
  <forward name="success" path="/catalog/Item.jsp" />
  <forward name="failure" path="/catalog/Product.jsp" />
</action>
```

第四步编辑 Item.jsp，现在开始发现一个问题，Item.jsp 中不只是显示 Item 信息，还有 Product 信息，而前面我们定义的是 Item 信息，如果使得 Item.jsp 显示 Product 信息呢，这就从设计起源 Domain Model 上考虑，在 Item 的 Model 中有 Product 引用：

```
private Product product;
public Product getProduct() { return product; }
public void setProduct(Product product) { this.product = product; }
```

Item 和 Product 的多对一关系其实应该在域建模开始就考虑到了。

那么，我们只要在持久层查询 Item 时，能够将其中的 Product 字段查询就可以。在持久层的 iBatis 的 Product.xml 实现有下列 SQL 语句：

```
<select id="getItem" resultMap="resultWithQuantity" parameterClass="string">
  select
    I.ITEMID, LISTPRICE, UNITCOST, SUPPLIER, I.PRODUCTID, NAME,
    DESCN, CATEGORY, STATUS, ATTR1, ATTR2, ATTR3, ATTR4, ATTR5, QTY
  from ITEM I, INVENTORY V, PRODUCT P where P.PRODUCTID = I.PRODUCTID and I.ITEMID
= V.ITEMID and I.ITEMID = #value#
</select>
```

这段语法实际在查询 Item 时，已经将 Product 查询出来，这样 Item Model 中已经有 Product 数据，因为 ActionForm 是 Model 映射，因此，前台 Jsp 也可以显示 Product 数据。

在 Item.jsp 中，进行下面定义：

```
<bean:define id="product" name="itemForm " property="product" />
<bean:define id="item" name="itemForm " />
```

将 itemForm 中 product 属性定义为 product 即可；这样不必大幅度修改原来的 Item.jsp 了。

整理模块配置

商品模块功能完成，struts 提供了多模块开发，因此我们可以将这一模块单独保存在一个配置中：/WEB-INF/struts-config-catalog.xml，这样以后扩展修改起来方便。

购物车模块实现

购物车属于一种有状态数据，也就是说，购物车的 scope 生命周期是用户，除非这个用户离开，否则购物车一直在内存中存在。

有态 POJO 服务

现在有两种解决方案：

第一，将购物车状态作为数据类，保存到 ActionForm 中，设置 scope 为 session，这种形式下，对购物车的数据操作如加入条目等实现不很方便，iBatis-jpetstore 4.0.5 就采取这个方案，在数据类 Cart 中存在大量数据操作方法，那么 Cart 这个类到底属于数据类 Model？还是属于处理服务类呢？

在我们 J2EE 编程中，通常使用两种类来实现功能，一种是数据类，也就是我们设计的 Model；一种是服务类，如 POJO 服务或 EJB 服务，服务属于一种处理器，处理过程。使用这两种分类比较方便我们来解析业务需求，EJB 中实体 Bean 和 Session Bean 也是属于这两种类型。

iBatis-jpetstore 4.0.5 则是将服务和数据类混合在一个类中，这也属于一种设计，但是我们认为它破坏了解决问题的规律性，而且造成数据和操作行为耦合性很强，在设计模式中我们还使用桥模式来分离抽象和行为，因此这种做法可以说是反模式的。那么我们采取数据类和服务分离的方式方案来试试看：

第二．购物车功能主要是对购物车这个 Model 的 crud，与通常的 crud 区别是，数据是保存到 HttpSession，而不是持久化到数据库中，是数据状态保存不同而已。所以如果我们实现一个 CartService，它提供 add 或 update 或 delete 等方法，只不过操作对象不是数据库，而是其属性为购物车 Cart，然后将该 CarService 实例保存到 HttpSession，实现每个用户一个 CartService 实例，这个我们成为有状态的 POJO 服务。

这种处理方式类似 EJB 架构处理，如果我们业务服务层使用 EJB，那么使用有态会话 Bean 实现这个功能。

现在问题是，Jdon 框架目前好像没有提供有状态 POJO 服务实例的获得，那么我们自己在 WebAppUtil.getService 获得实例后，保存到 HttpSession 中，下次再到 HttpSession 中获得，这种有状态处理需要表现层更多代码，这就不能使用 Jdon 框架的 crud 配置实现了，需要我们代码实现 ModelHandler 子类。

考虑到可能在其他应用系统还有这种需求，那么能不能将有状态的 POJO 服务提炼到 Jdon 框架中呢？关键使用什么方式加入框架，因为这是设计目标服务实例的获得，框架主

要流程代码又不能修改，怎么办？

Jdon 框架的 AOP 功能在这里显示了强大灵活性，我们可以将有状态的 POJO 服务实例获得作为一个拦截器，拦截在原来 POJO 服务实例获得之前。在 Jdon 框架设计中，目标服务实例的获得一般只有一次。

创建有状态 POJO 服务拦截器 com.jdon.aop.interceptor. StatefulInterceptor，再创建一个空接口：com.jdon.controller.service.StatefulPOJOService，需要实现有状态实例的 POJO 类只要继承这个接口就可以。

配置 aspect.xml，加入这个拦截器：

```
<interceptor name="statefulInterceptor" class="com.jdon.aop.interceptor.StatefulInterceptor"
        pointcut="pojoServices" />
```

这里需要注意的是：你不能让一个 POJO 服务类同时继承 Poolable，然后又继承 Stateful，因为这是两种不同的类型，前者适合无状态 POJO；后者适合 CartService 这样有状态处理；这种选择和 EJB 的有态/无态选择是一样的。

Model 和 Service 设计

购物车模块主要围绕域模型 Cart 展开，需要首先明确 Cart 是一个什么样的业务模型，购物车页面是类似商品条目批量查询页面，不过购物车中显示的不但是商品条目，还有数量，那么我们专门创建一个 Model 来指代它，取名为 CartItem，CartItem 是 Item 父集，多了一个数量。

这样购物车页面就是 CartItem 的批量查询页面，然后还有 CartItem 的 crud 操作，所以购物车功能主要是 CartItem 的 crud 和批量查询功能。

iBatis 4.0.5 原来设计了专门 Cart Model，其实这个 Cart 主要是一个功能类，因为它的数据项只有一个 Map 和 List，这根本不能代表业务需求中的一个模型。虽然 iBatis 4..0.5 也可以自圆其说实现了购物车功能，但是这种实现是随心所欲，无规律性可遵循，因而以后维护起来也是困难，维护人员理解困难，修改起来也没有章程可循，甚至乱改一气。

CartItem 可以使用 iBatis 原来的 CartItem，这样也可保持 Cart.jsp 页面修改量降低。删除原来的 Cart 这个 Model，建立对应的 CartService，实现原来的 Cart 一些功能。

```
public interface CartService {
        CartItem getCartItem(String itemId);
        void addCartItem(EventModel em);
        void updateCartItem(EventModel em);
        void deleteCartItem(EventModel em);
        PageIterator getCartItems();
}
```

CartServiceImp 是 CartService 子类，它是一个有状态 POJO 服务，代码简要如下：

```
public class CartServiceImp implements CartService, Stateful{
    private ProductManager productManager;
    //将原来 iBatis 中 Cart 类中两个属性移植到 CartServiceImp 中
    private final Map itemMap = Collections.synchronizedMap(new HashMap());
    private   List itemList = new ArrayList();

    public CartServiceImp(ProductManager productManager) {
        super();
        this.productManager = productManager;
    }
    ……
```

```
}
```

itemMap 是装载 CartItem 的一个 Map，是类属性，由于 CartServiceImp 是有状态的，每个用户一个实例，那么也就是每个用户有自己的 itemMap 列表，也就是购物车。

CartServiceImp 中的 getCartItemIDs 是查询购物车当前页面的购物条目，属于批量分页查询实现，这里有一个需要考量的地方，是 getCartItems 方法还是 getCartItemIDs 方法？也就是返回 CartIem 的实例集合还是 CartItem 的 ItemId 集合？按照前面标准的 Jdon 框架批量分页查询实现，应该返回 CartItem 的 ItemId 集合，然后由 Jdon 框架的 ModelListAction 根据 ItemId 首先从缓存中获得 CartItem 实例，但是本例 CartItem 本身不是持久化在数据库，而也是内存 HttpSession 中，所以 ModelListAction 这种流程似乎没有必要。

如果将来业务需求变化，购物车状态不是保存在内存而是数据库，这样，用户下次登陆时，可以知道他上次购物车里的商品条目，那么采取 Jdon 框架标准查询方案还是有一定扩展性的。

这里，我们就事论事，采取返回 CartIem 的实例集合，展示一下如何灵活应用 Jdon 框架的批量查询功能。下面是 CartServiceImp 的 getCartItems 方法详细代码：

```
 public PageIterator getCartItems(int start, int count) {
            int offset = itemList.size() - start; //获得未显示的总个数
            int pageCount = (count < offset)?count:offset;
            List pageList = new ArrayList(pageCount); //当前页面记录集合
            for(int i=start; i< pageCount + start;i++){
                  pageList.add(itemList.get(i));
            }
            int allCount = itemList.size();
            int currentCount = start + pageCount;
            return new PageIterator(allCount, pageList.toArray(new CartItem[0]), start,
                        (currentCount < allCount)?true:false);
}
```

getCartItems 方法是从购物车所有条目 itemList 中查询获得当前页面的条目，并创建一个 PageIterator。

注意，现在这个 PageIterator 中 keys 属性中装载的不是数据 ID 集合，而是完整的 CartItem 集合，因为上面代码中 pageList 中对象是从 itemList 中获得，而 itemList 中装载的都是 CartItem。

表现层购物车显示功能

由于 PageIterator 中封装的是完整 Model 集合，而不是 ID 集合，所以现在表现层有两种方案，继承框架的 ModelListAction；或重新自己实现一个 Action，替代 ModelListAction。

这里使用继承框架的 ModelListAction 方案，巧妙地实现我们的目的，省却编码：

```
public class CartListAction extends ModelListAction {

    public PageIterator getPageIterator(HttpServletRequest request, int arg1, int arg2) {
        CartService cartService = (CartService)WebAppUtil.getService("cartService", request);
        return cartService.getCartItems();
    }

    public Model findModelByKey(HttpServletRequest arg0, Object key) {
        return (Model)key; //因为 key 不是主键，而是完整的 Model，直接返回
    }
```

```
        protected boolean isEnableCache(){
            return false;   //无需缓存，CartItem 本身实际是在内存中。
        }

    }
```

配置 struts-config.xml：

```
<form-beans>
    <form-bean name="listForm" type="com.jdon.strutsutil.ModelListForm"/>
</form-beans>
 <action-mappings>
      <action path="/shop/viewCart"
                type="com.jdon.framework.samples.jpetstore.presentation.action.CartListAction"
        name="listForm" scope="request"
        validate="false">
        <forward name="success" path="/cart/Cart.jsp"/>
      </action>
      ……
  </action-mappings>
```

上面是购物车显示实现，只要调用/shop/viewCart.shtml 就可以显示购物车了。

在 Cart.jsp 页面插入下面标签：

```
<logic:iterate id="cartItem" name="listForm" property="list">
      ….
</logic:iterate>
```

分页显示标签如下：

```
<MultiPages:pager actionFormName="listForm" page="/shop/viewCart.shtml">
<MultiPages:prev name="<font color=green><B>&lt;&lt; Prev</B></font>"/>
<MultiPages:index />
<MultiPages:next name="<font color=green><B>Next &gt;&gt;</B></font>"/>
</MultiPages:pager>
```

购物车新增删除条目功能

前面完成了购物车显示功能，下面是设计购物车的新增和删除、修改功能。

参考 Jdon 框架的 crud 功能实现，Model 是 CartItem，配置 jdonframework.xml 使其完成新增删除功能：

```
        <model key="workingItemId"
            class="com.jdon.framework.samples.jpetstore.domain.CartItem">
            <actionForm name="cartItemForm"/>
            <handler>
                <service ref="cartService">
                    <createMethod name="addCartItem"/>
                    <deleteMethod name="deleteCartItem"/>
                </service>
            </handler>
        </model>
```

在这个配置中，只有新增和删除方法，修改方法没有，因为购物车修改主要是其中商品

条目的数量修改，它不是逐条修改，而是一次性批量修改，这里的 Model 是 CartItem，这是购物车里的一个条目，因此如果这里写修改，也只是 CartItem 一个条目的修改，不符合我们要求。下面专门章节实现这个修改。

表现层主要是配置，没有代码，代码都依靠 cartService 中的 addCartItem 和 deleteCartItem 实现：例如：

```
public void addCartItem(EventModel em) {
    CartItem cartItem = (CartItem) em.getModel();
    String workingItemId = cartItem.getWorkingItemId();
    ……
}
```

注意 addCartItem 中从 EventModel 实例中获取的 Model 是 CartItem，这与我们在 jdonframework.xml 中上述定义的 Model 类型是统一的。

Struts-config.xml 中定义是 crud 的标准定义，注意，这里只有 ModelSaveAction 无需 ModelViewAction，因为将商品条目加入或删除购物车这个功能没有专门的显示页面：

```
<action path="/shop/addItemToCart" type="com.jdon.strutsutil.ModelSaveAction"
  name="cartItemForm" scope="request"
  validate="false">
  <forward name="success" path="/cart/Cart.jsp"/>
</action>

<action path="/shop/removeItemFromCart" type="com.jdon.strutsutil.ModelSaveAction"
  name="cartItemForm" scope="request"
  validate="false">
  <forward name="success" path="/cart/Cart.jsp"/>
</action>
```

注意，调用删除功能时，需要附加 action 参数：

/shop/removeItemFromCart.shtml?action=delete

而/shop/addItemToCart.shtml 是新增属性，缺省后面无需跟参数。

购物车条目批量修改功能

上面基本完成了购物车主要功能;购物车功能一个复杂性在于其显示功能和修改功能合并在一起，修改功能是指修改购物车里所有商品条目的数量。

既然有修改功能，而且这个修改功能比较特殊，我们需要设计一个独立的 ActionForm，用来实现商品条目数量的批量修改。

首先设计一个 ActionForm（ModelForm），该 ModelForm 主要用来实现购物车条目数量的更改，取名为 CartItemsForm，其内容如下：

```
public class CartItemsForm extends ModelForm {
    private String[] itemId;
    private int[]quantity;
    private BigDecimal totalCost;
    …..
}
```

itemId 和 quantity 设计成数组，这样，Jsp 页面可以一次性提交多个 itemId 和 quantity 数值。

现在 CartItemsForm 已经包含前台 jsp 输入的数据，我们还是将其传递递交到服务层实现处理。因此建立一个 Model，内容与 CartItemsForm 类似，这里的 Model 名为 CartItems，

实际是一个传输对象。

```
public class CartItems extends Model{
    private String[] itemId;
    private int[] quantity;
    private BigDecimal totalCost;
    ……
}
```

表现层在 jdonframework.xml 定义配置就无需编码，配置如下：

```
<model key=" "
    class="com.jdon.framework.samples.jpetstore.domain.CartItems">
    <actionForm name="cartItemsForm"/>
    <handler>
        <service ref="cartService">
            <updateMethod name="updateCartItems"/>
        </service>
    </handler>
</model>
```

上面配置中，Model 是 CartItems，ActionForm 是 cartItemsForm，这两个是专门为批量修改设立的。只有一个方法 updateMethod。因为在这个更新功能中，没有根据主键从数据库查询 Model 的功能，因此，这里 model 的 key 可以为空值。

服务层 CartServiceImp 的 updateCartItems 方法实现购物车条目数量更新：

```
public void updateCartItems(EventModel em) {
    CartItems cartItems = (CartItems) em.getModel();
    try {
        String[] itemIds = cartItems.getItemId();
        int[] qtys = cartItems.getQuantity();
        int length = itemIds.length;
        for (int i = 0; i < length; i++) {
            updateCartItem(itemIds[i], qtys[i]);//逐条更新购物车中的数量
        }
    } catch (Exception ex) {
        logger.error(ex);
    }
}
```

注意 updateCartItems 中从 EventModel 取出的是 CartItems，和前面 addCartItem 方法中取出的是 CartItem Model 类型不一样，这是因为这里我们在 jdonframework.xml 中定义与 updateCartItems 相对应的 Model 是 CartItems.

最后一步工作是 Cat.jsp 中加入 CartItemsForm，能够在购物车显示页面有一个表单提交，客户按提交按钮，能够立即实现当前页面购物车数量的批量修改。Cat.jsp 加入如下代码：

```
<html:form action="/shop/updateCartQuantities.shtml" method="post" >
<html:hidden property="action" value="edit" />
……
<input type="hidden" name="itemId" value="<bean:write name="cartItem" property="workingItemId"/>">
    <input type="text" size="3" name="quantity" value="<bean:write name="cartItem"
                                                    property="quantity"/>" />
…….
```

注意，一定要有 action 赋值 edit 这一行，这样提交给 updateCartQuantities.shtml 实际是

ModelSaveAction 时，框架才知道操作性质。

购物车总价显示功能

最后，还有一个功能需要完成，在购物车显示时，需要显示当前购物车的总价格，注意不是显示当前页面的总价格，所以无法在 Cart.jsp 直接实现，必须遍历购物车里所有 CartItem 计算总数。

该功能是购物车显示时一起实现，购物车显示是通过 CartListAction 实现的，这个 CartListAction 实际是生成一个 ModelListForm，如果 ModelListForm 能够增加一个 getTotalPrice 方法就可以，因此有两种实现方式：继承 ModelListForm 加入自己的 getTotalPrice 方法；第二种无需再实现自己的 ModelListForm，ModelListForm 可以携带一个 Model，通过 setOneModel 即可，这个方法是在 ModelListAction 的子类 CartListAction 可以 override 覆盖实现的，在 CartListAction 加入下列方法：

```
protected Model setOneModel(HttpServletRequest request){
        CartService cartService = (CartService)WebAppUtil.getService("cartService", request);
        CartItems cartItems = new CartItems();
        cartItems.setTotalCost(cartService.getSubTotal());
        return cartItems;
    }
```

我们使用空的 CartItems 作为携带价格总数的 Model，然后在 Cart.jsp 中再取出来显示：

```
<bean:define id="cartItems " name="listForm" property="oneModel" />
<b>Sub Total: <bean:write name="cartItems" property="subTotal" format="$#,##0.00" />
```

将当前页面 listForm 中属性 oneModel 定义为 cartItems，它实际是我们定义的 CartItems，下一行取出总价即可。

用户喜欢商品列表功能

在显示购物车时，需要一起显示该用户喜欢的商品列表，很显然这是一个批量分页查询实现，但是它有些特殊，它首先显示的第一页不是由 URL 调用的，而是嵌入在购物车显示中，那么只能在购物车显示页面的 ModellistForm 中做文章。

在上节中，在 CartListAction 中 setOneModel 方法中，使用 CartItems 作为价格总数的载体，现在恐怕我们也要将之作为本功能实现载体。

还有一种实现载体，就是其他 scope 为 session 的 ActionForm，AccountForm 很适合做这样的载体，而且和本功能意义非常吻合，所以在 AccountForm/Account 中增加一个 myList 字段，在 myList 字段中，放置的是该用户喜欢的商品 Product 集合，注意不必放置 Product 的主键集合，因为我们只要显示用户喜欢商品的第一页，这一页是嵌入购物车显示页面中，所以第一页显示的个数是由程序员可事先在程序中定义。

这样在 Account 获得时，一起将 myList 集合值获得。

订单模块实现

我们还是从域模型开始，Order 是订单模块的核心实体，其内容可以确定如下：

```
public class Order extends Model {

  /* Private Fields */
```

```
    private int orderId;
    private String username;
    private Date orderDate;
    private String shipAddress1;
    private String shipAddress2;
    …..
}
```

第二步，建立与 Model 对应的 ModelForm，我们可以称之为边界模型，代码从 Order 拷贝过来即可。当然 OrderForm 还有一些特殊的字段以及初始化：

```
public class OrderForm extends ModelForm
    private boolean shippingAddressRequired;
    private boolean confirmed;
    static {
        List cardList = new ArrayList();
        cardList.add("Visa");
        cardList.add("MasterCard");
        cardList.add("American Express");
        CARD_TYPE_LIST = Collections.unmodifiableList(cardList);
      }

    public OrderForm(){
        this.shippingAddressRequired = false;
        this.confirmed = false;
    }
   …..
}
```

第三步，建立 Order Model 的业务服务接口，如下：

```
public interface OrderService {
     void insertOrder(Order order);
     Order getOrder(int orderId);
     List getOrdersByUsername(String username);
}
```

第四步，实现 OrderService 的 POJO 子类：OrderServiceImp。

第五步，表现层实现，本步骤可和第四步同时进行。

OrderService 中有订单的插入创建功能，我们使用 Jdon 框架的 crud 中 create 配置实现，配置 struts-config.xml 和 jdonframework.xml：

```
        <form-bean name="orderForm"
                type="com.jdon.framework.samples.jpetstore.presentation.form.OrderForm"/>
```

```
<model key="orderId"
        class="com.jdon.framework.samples.jpetstore.domain.Order">
        <actionForm name="orderForm"/>
        <handler>
            <service ref="orderService">
                <createMethod name="insertOrder"/>
            </service>
        </handler>
</model>
```

第六步：根据逐个实现界面功能，订单的第一个功能创建一个新的订单，在新订单页面 NewOrderForm.jsp 推出之前，这个页面的 ActionForm 已经被初始化，是根据购物车等 Cart 其他 Model 数据初始化合成的。

新订单页面初始化

根据 Jdon 框架中 crud 功能实现，初始化一个 ActionForm 有两种方法：一继承 ModelHandler 实现 initForm 方法；第二通过 jdonframework.xml 的 initMethod 方法配置。

这两个方案选择依据是根据用来初始化的数据来源什么地方。

订单表单初始化实际是来自购物车信息或用户账号信息，这两个都事先保存在 HttpSession 中，购物车信息是通过有态 CartService 实现的，所以这些数据来源是和 request 相关，那么我们选择第一个方案。

继承 ModelHandler 之前，我们可以考虑首先继承 ModelHandler 的子类 XmlModelHandler，只要继承 initForm 一个方法即可，这样节省其他方法编写实现。

```
public class OrderHandler extends XmlModelHandler {

    public ModelForm initForm(HttpServletRequest request) throws
    Exception{
        HttpSession session = request.getSession();
        AccountForm accountForm = (AccountForm) session.getAttribute("accountForm");
        OrderForm orderForm = createOrderForm(accountForm);
        CartService cartService = (CartService)WebAppUtil.getService("cartService", request);

        orderForm.setTotalPrice(cartService.getSubTotal());

        //below can read from the user's creditCard service;
        orderForm.setCreditCard("999 9999 9999 9999");
        orderForm.setExpiryDate("12/03");
        orderForm.setCardType("Visa");
        orderForm.setCourier("UPS");
        orderForm.setLocale("CA");
        orderForm.setStatus("P");

        Iterator i = cartService.getAllCartItems().iterator();
        while (i.hasNext()) {
           CartItem cartItem = (CartItem) i.next();
           orderForm.addLineItem(cartItem);
        }
        return orderForm;
    }

    private OrderForm createOrderForm(AccountForm account){
       ……
    }
}
```

ModelHandler 的 initForm 继承后，因为这使用了 Jdon 的 crud 功能实现，这里我们只使用到 crud 中的创建功能，因此，findModelBykey 方法就无需实现，或者可以在 jdonframework.xml 中配置该方法实现。

考虑到在 initForm 执行后，需要推出一个 NewOrderForm.jsp 页面，这是一个新增性质的页面。所以在 struts-config.xml

```
<action path="/shop/newOrderForm" type="com.jdon.strutsutil.ModelViewAction"
  name="orderForm" scope="request"       validate="false">
      <forward name="create" path="/order/NewOrderForm.jsp"/>
</action>
```

订单确认流程

新的订单页面推出后，用户需要经过两个流程才能确认保存，这两个流程是填写送货地址以及再次完整确认。这两个流程实现的动作非常简单，就是将 OrderForm 中的 shippingAddressRequired 字段和 confirm 字段赋值，相当于简单的开关，这是一个很简单的动作，可以有两种方案：直接在 jsp 表单中将这两个值赋值；直接使用 struts 的 Action 实现。后者需要编码，而且不是非有这个必要，只有第一个方案行不通时才被迫实现。

第一步：填写送货地址

使用 Jdon 框架的推出纯 Jsp 功能的 Action 配置 struts-config.xml：

```
<action path="/shop/shippingForm" type="com.jdon.strutsutil.ForwardAction"
  name="orderForm" scope="session"          validate="false">
  <forward name="forward" path="/order/ShippingForm.jsp"/>
</action>
```

这是实现送货地址页面的填写，使用的还是 OrderForm。更改前面流程 NewOrderForm.jsp 中的表单提交 action 值为本 action path: shippingForm.shtml：

```
<html:form action="/shop/shippingForm.shtml" styleId="orderForm" method="post" >
  ……
</html:form>
```

在 ShippingForm.jsp 中增加将 shippingAddressRequired 赋值的字段：

```
<html:hidden name="orderForm"    property="shippingAddressRequired" value="false"/>
```

第二步：确认订单

类似上述步骤，配置 struts-config.xml：

```
<action path="/shop/confirmOrderForm" type="com.jdon.strutsutil. ForwardAction"
  name="orderForm" scope="session"    validate="false">
  <forward name="forward" path="/order/ConfirmOrder.jsp"/>
</action>
```

将上一步 ShippingForm.jsp 的表单 action 改为本 action 的 path: confirmOrderForm.shtml：

```
<html:form action="/shop/confirmOrderForm.shtml" styleId="orderBean" method="post" >
```

修改 ConfirmOrder.jsp 中提交的表单为最后一步，保存订单 newOrder.shtml：

```
<html:link page="/shop/newOrder.shtml?confirmed=true"><img border="0"
src="../images/button_continue.gif" /></html:link>
```

第三步：下面是创建数据保存功能实现：

```
<action path="/shop/newOrder" type="com.jdon.strutsutil.ModelSaveAction"
  name="orderForm" scope="session"
  validate="true" input="/order/NewOrderForm.jsp">
  <forward name="success" path="/order/ViewOrder.jsp"/>
</action>
```

ModelSaveAction 是委托 ModelHandler 实现的，这里有两种方式：配置方式：在 jdonframework.xml 中配置了方法插入；第二种是实现代码，这里原本可以使用配置方式实现，但是因为在功能上有要求：在订单保存后，需要清除购物车数据，因此只能使用代码实

现方式，在 ModelHandler 中实现子类方法 serviceAction：

```
public void serviceAction(EventModel em, HttpServletRequest request) throws java.lang.Exception {
    try {
        CartService cartService = (CartService) WebAppUtil.getService("cartService", request);
        cartService.clear(); //清楚购物车数据

        OrderService orderService = (OrderService) WebAppUtil.getEJBService("orderService", request);
        switch (em.getActionType()) {
                case Event.CREATE:
                    Order order = (Order) em.getModel();
                    orderService.insertOrder(order);
                    cartService.clear();
                    break;
            case Event.EDIT:
                    break;
            case Event.DELETE:
                    break;
          }
    } catch (Exception ex) {
                throw new Exception(" serviceAction Error:" + ex);
    }
}
```

用户订单列表

用户查询自己的订单列表功能很明显可以使用 Jdon 框架的批量查询事先。
在 struts-config.xml 中配置 ModelListForm 如下：

```
        <form-bean name="listForm" type="com.jdon.strutsutil.ModelListForm"/>
```

建立继承 ModelListAction 子类 OrderListAction：

```
public class OrderListAction extends ModelListAction {

    public PageIterator getPageIterator(HttpServletRequest request, int start, int count) {
        OrderService orderService = (OrderService) WebAppUtil.getService("orderService", request);
        HttpSession session = request.getSession();
        AccountForm accountForm = (AccountForm) session.getAttribute("accountForm");
        if (accountForm == null) return new PageIterator();
        return orderService.getOrdersByUsername(accountForm.getUsername(), start, count);
    }

    public Model findModelByKey(HttpServletRequest request, Object key) {
        OrderService orderService = (OrderService) WebAppUtil.getService("orderService", request);
        return orderService.getOrder((Integer)key);
    }

}
```

修改 OrderService，将获得 Order 集合方法改为：

```
public class OrderService{

    PageIterator getOrdersByUsername(String username, int start, int count)
```

```
        ....
    }
```

根据 Jdon 批量查询要求，使用 iBatis 实现返回 ID 集合以及符合条件的总数。

最后编写 ListOrders.jsp，两个语法：logic:iterator 和 MultiPages

配置 jdon 框架启动

目前我们有四个 struts-config.xml，前面每个模块一个配置：

/WEB-INF/struts-config.xml 主配置

/WEB-INF/struts-config-catalog.xml 商品相关配置

/WEB-INF/struts-config-security.xml 用户相关配置

/WEB-INF/struts-config-cart.xml 购物车相关配置

/WEB-INF/struts-config-order.xml 订单相关配置

本项目只有一个 jdonframework.xml，当然我们也可以创建多个 jdonframework.xml，然后在其 struts-config.xml 中配置。

```
    <plug-in className="com.jdon.strutsutil.InitPlugIn">
      <set-property property="modelmapping-config"    value="jdonframework_iBATIS.xml" />
    </plug-in>
```

修改 iBatis 的 DAO 配置

iBatis 4.0.5 中原来的配置过于扩张（从持久层扩张到业务层），AccountDao 每个实例获得都需要通过 daoManager.getDao 这样形式，而使用 Jdon 框架后，AccountDao 等 DAO 实例获得无需特别语句，我们只要在 AccountService 直接引用 AccountDao 接口，至于 AccountDao 的具体实例，通过 Ioc 注射进入即可。

因此，在 jdonframework.xml 中有如下配置：

```
<pojoService name="accountDao"
        class="com.jdon.framework.samples.jpetstore.persistence.dao.sqlmapdao.AccountSqlMapDao"/>
<pojoService name="accountService"
            class="com.jdon.framework.samples.jpetstore.service.bo.AccountServiceImp"/>
<pojoService name="productManager"
            class="com.jdon.framework.samples.jpetstore.service.bo.ProductManagerImp"/>
```

而 AccountServiceImp 代码如下：

```
public class AccountServiceImp implements AccountService, Poolable {
    private AccountDao accountDao;
    private ProductManager productManager;

    public AccountServiceImp(AccountDao accountDao,
                             ProductManager productManager){
        this.accountDao = accountDao;
        this.productManager = productManager;
    }
```

AccountServiceImp 需要两个构造方法实例，这两个中有一个是 AccountDao。

按照 iBatis 原来的 AccountDao 子类 AccountSqlMapDao 有一个构造方法参数是

DaoManager，但是我们无法生成自己的 DaoManager 实例，因为 DaoManager 是由 dao.xml 配置文件读取后生成的，这是一个动态实例；那只有更改 AccountSqlMapDao 构造方法了。

根源在于 BaseSqlMapDao 类，BaseSqlMapDao 是一个类似 JDBC 模板类，每个 Dao 都继承它，现在我们修改 BaseSqlMapDao 如下：

```
public class BaseSqlMapDao extends DaoTemplate implements SqlMapExecutor{
      …..
}
```

BaseSqlMapDao 是 XML 配置和 JDBC 模板的结合体，在这个类中，这两者搭配在一起，在其中实现 SqlMapExecutor 各个子方法。

我们再创建一个 DaoManagerFactory，专门根据配置文件创建 DaoManager 实例：

主要方法如下：

```
Reader reader = Resources.getResourceAsReader(daoResource);
daoManager = DaoManagerBuilder.buildDaoManager(reader);
```

其中 daoResource 是 dao.xml 配置文件，这个配置是在 jdonframework.xml 中配置：

```
<pojoService name="daoManagerFactory"
            class="com.jdon.framework.samples.jpetstore.persistence.dao.DaoManagerFactory">
            <constructor
              value="com/jdon/framework/samples/jpetstore/persistence/dao/sqlmapdao/sql/dao.xml"/>
</pojoService>
```

这样，我们可以通过改变 jdonframework.xml 配置改变 dao.xml 配置。

Dao.xml 配置如下：

```
<daoConfig>
  <context>
    <transactionManager type="SQLMAP">
      <property name="SqlMapConfigResource"
     value="com/jdon/framework/samples/jpetstore/persistence/dao/sqlmapdao/sql/sql-map-config.xml"/>
    </transactionManager>

    <dao interface="com.ibatis.sqlmap.client.SqlMapExecutor"
   implementation="com.jdon.framework.samples.jpetstore.persistence.dao.sqlmapdao.BaseSqlMapDao"/>

  </context>
</daoConfig>
```

在 dao.xml 中，我们只配置一个 JDBC 模板，而不是将所有的如 AccountDao 配置其中，因为我们需要 iBatis 只是它的 JDBC 模板，实现持久层方便的持久化，仅此而已！

DaoManagerFactory 为我们生产了 DaoManager 实例，那么如何赋值到 BaseSqlMapDao 中，我们设计一个创建 BaseSqlMapDao 工厂如下：

```
public class SqlMapDaoTemplateFactory {

    private DaoManagerFactory daoManagerFactory;

    public SqlMapDaoTemplateFactory(DaoManagerFactory daoManagerFactory) {
        this.daoManagerFactory = daoManagerFactory;
    }

    public SqlMapExecutor getSqlMapDaoTemp(){
        DaoManager daoManager = daoManagerFactory.getDaomanager();
```

```
            return (SqlMapExecutor)daoManager.getDao(SqlMapExecutor.class);
        }

    }
```

通过 getSqlMapDaoTemp 方法，由 DaoManager.getDao 方法获得 BaseSqlMapDao 实例，BaseSqlMapDao 的接口是 SqlMapExecutor，这样我们通过 DaoManager 获得一个 JDBC 模板 SqlMapExecutor 的实例。

这样，在 AccountDao 各个子类 AccountSqlMapDao 中，我们只要通过 SqlMapDaoTemplateFactory 获得 SqlMapExecutor 实例，委托 SqlMapExecutor 实现 JDBC 操作，如下：

```
public class AccountSqlMapDao implements AccountDao {
  //iBatis 数据库操作模板
  private SqlMapExecutor sqlMapDaoTemplate;
    //构造方法
  public AccountSqlMapDao(SqlMapDaoTemplateFactory sqlMapDaoTemplateFactory) {
      sqlMapDaoTemplate = sqlMapDaoTemplateFactory.getSqlMapDaoTemp();
  }
  //查询数据库
  public Account getAccount(String username) throws SQLException{
    return (Account)sqlMapDaoTemplate.queryForObject("getAccountByUsername", username);
  }
```

部署调试

使用 Jdon 框架开发 Jpetstore， 一次性调试通过率高，一般问题都是存在数据库访问是否正常，一旦正常，主要页面就出来了，其中常见问题是 jsp 页面和 ActionForm 的字段不对应，如 jsp 页面显示如下错误：

No getter method available for property creditCardTypes for bean under name orderForm

表示在 OrderForm 中没有字段 creditCardTypes，或者有此字段，但是大小写错误等粗心问题。

如果 jsp 页面或后台 log 记录显示：

System error! please call system Admin.java.lang.Exception

一般这是由于前面出错导致，根据记录向前搜索，搜索到第一个出错记录：

2005-07-07 11:55:16,671 [http-8080-Processor25] DEBUG com.jdon.container.pico.PicoContainerWrapper - getComponentClass: name=orderService

2005-07-07 11:55:16,671 [http-8080-Processor25] ERROR com.jdon.aop.reflection.MethodConstructor - no this method name:insertOrder

第一个出错是在 MethodConstructor 报错，没有 insertOrder 方法，根据上面一行是 orderService，那么检查 orderService 代码看看有无 insertOrder：

```
public interface OrderService {
      void insertOrder(Order order);
      Order getOrder(int orderId);
      List getOrdersByUsername(String username);
}
```

OrderService 接口中是有 insertOrder 方法，那么为什么报错没有呢？仔细检查一下，是不是 insertOrder 的方法参数有问题，哦， 因为 orderService 的调用是通过 jdonframework.xml 下面配置进行的：

```
<model key="orderId"
        class="com.jdon.framework.samples.jpetstore.domain.Order">
        <actionForm name="orderForm"/>
        <handler>
            <service ref="orderService">
                <createMethod name="insertOrder"/>
            </service>
        </handler>
    </model>
```

而根据 Jdon 框架要求，使用配置实现 ModelHandler，则要求 OrderService 的 insertOrder 方法参数必须是 EventModel，更改 OrderService 的 insertOrder 方法如下：

```
public interface OrderService {
    void insertOrder(EventModel em);
}
```

同时，修改 OrderService 的子类代码 OrderServiceImp：

```
public void insertOrder(EventModel em) {
        Order order = (Order)em.getModel();
        try{
            orderDao.insertOrder(order);
        }catch(Exception daoe){
            Debug.logError(" Dao error : " + daoe, module);
            em.setErrors("db.error");
        }
    }
```

注意 em.setErrors 方法，该方法可向 struts 页面显示出错信息，db.error 是在本项目的 application.properties 中配置的。

关于本次页面出错问题，还有更深缘由，因为我们在 jdonframework.xml 中中定义了 createMethod，而根据前面已经有 ModelHandler 子类代码 OrderHandler 实现，所以，这里不用配置实现，jdonframework.xml 的配置应该如下：

```
    <model key="orderId"
        class="com.jdon.framework.samples.jpetstore.domain.Order">
        <actionForm name="orderForm"/>
        <handler class="com.jdon.framework.samples.jpetstore.presentation.action.OrderHandler"/>
    </model>
```

直接定义了 hanlder 的 class 是 OrderHandler，在 OrderHandler 中的 serviceAction 我们使用代码调用了 OrderService 的 insertOrder 方法，如果使用这样代码调用，无需要求 OrderService 的 insertOrder 的参数是 EventModel 了。

总结

Jpetstore 整个开发大部分基于 Jdon 框架开发，特别是表现层，很少直接接触使用 struts

原来功能，Jdon 框架的表现层架构基本让程序员远离了 struts 的烦琐开发过程，又保证了 struts 的 MVC 实现。

Jpetstore 中只有 SignonAction 这个类是直接继承 struts 的 DispatchAction，这个功能如果使用基于 J2EE 容器的安全认证实现（见 JdonNews），那么 Jpetstore 全部没有用到 struts 的 Action，无需编写 Action 代码；ActionForm 又都是 Model 的拷贝，Action 和 ActionForm 是 struts 编码的两个主要部分，这两个部分被 Jdon 框架节省后，整个 J2EE 的 Web 层开发方便快速，而且容易得多。

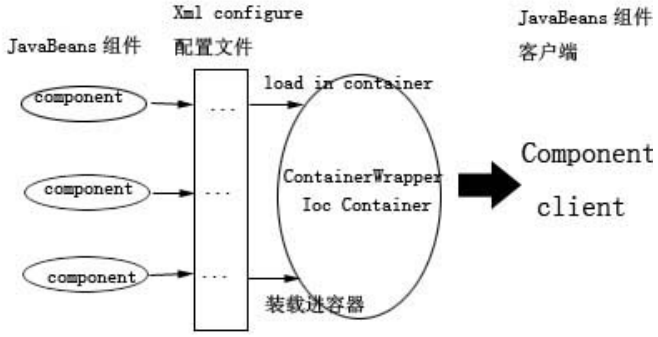
这说明 Jdon 框架确实是一款快速开发 J2EE 工具，而且是非常轻量的。

纵观 Jpetstore 系统，主要有三个层的配置文件组成，持久层由 iBatis 的 Product.xml 等配置文件组成；服务层由 jdon 框架的 jdonframework.xml 组成；表现层由 struts 的 struts-config.xml 和 jdonframework.xml 组成；剩余代码基本是 Model 之类实现。

Jdon 框架核心设计架构

架构设计

由于整个框架中中组件基于 Ioc 实现，组件类之间基本达到完全解耦。



从上图中可以看出，任何 JavaBeans 组件只要在 XML 配置文件（container.xml aspect.xml 和 jdonframework.xml）中配置，由容器装载机制注册到 Ioc 容器中，这些组件的客户端，也就是访问者，只要和微容器（或者组件的接口）打交道，就可以实现组件访问。

因此，本框架中每个功能块都可从框架中分解出来，单独使用它们。用户自己的任意功能块也可以加入框架中，Jdon 框架成为一种完全开放、伸缩自如的基础平台。

包结构

Jdon 框架有下列包名组成：

AOP ：表示 AOP 相关功能的类，其与 bussinessproxy 包关系最紧密，两者是系统的核心包。

Bussinessproxy：与动态代理、目标服务相关的类。

Container： 实现将组件装载入容器，这是与 Ioc 微容器相关的类，缺省使用了 PicoContainer，但是可以替换的。包括容器的组件注册、配置读取、组件访问等功能。以上三个包可从 Jdon 框架中独立出来。

Controller: 这是一些基础功能类、以及和应用相关类的功能包，该包是 Container 的客户端。

Model：这是和数据模型增删改查（crud）。批量查询、缓存优化等相关功能的类。

Security：这是和用户注册登陆相关功能类，

Servicelocator：这是和 EJB 运行定位相关的类

Strutsutil：与 Struts 相关类，前面 Model 包中各种功能必须通过具体表现层技术实现。

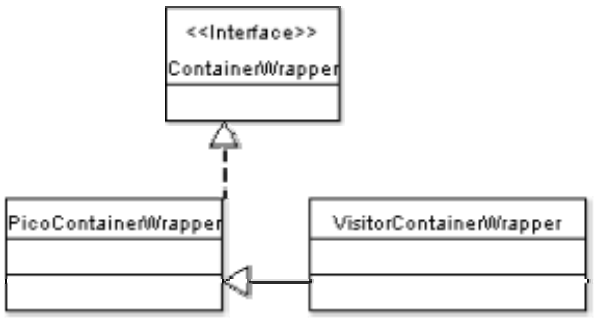
Util：一些工具类。

container 包

Container 包主要是负责容器管理方面的功能，其他包中的组件都被写入配置文件 container.xml、aspect.xml 和 jdonframework.xml 中，而 container 包主要负责与这些配置文件打交道，从配置文件中获得其他包的组件，向容器中注册，并启动容器。

主要一个接口是 ContainerWrapper， ContainerWrapper 有两个主要方法：向容器注册组件；从容器查询获得组件。

ContainerWrapper 接口的缺省实现是 PicoContainerWrapper 和 VisitorContainerWrapper 两个子类，如下图：



PicoContainerWrapper 使用了著名的 PicoContainer 实现（http://www.picocontainer.org），ContainerWrapper 接口主要从其抽象出来。

访问者模式

VisitorContainerWraper 是观察者模式的实现，传统观察者模式中，对于 Visitor 角色一般有下面多个访问不同被访问者的方法：

visitAcomponent();

visitBcomponent();

…..

由于 Acomponent、Bcomponent 这些类已经注册到容器，因此，通过容器可以直接直接实现不同组件的访问，只需通过下面一个方法实现：

```
public ComponentVisitor getComponentVisitor(){
        return new ComponentOriginalVisitor(this);
}
```

而访问者 Visitor 则可以节省为单一方法即可：

visit(XXX xxx);

使用访问者模式的原因：主要为了实现缓存，提高一些组件运行性能，如果一些组件每次访问时，都需要 new，例如 Proxy.newInstance 如果频繁反复运行，将是十分耗费性能的，因此，使用缓存尽量避免每次创建将提高系统的运行性能。

Visitor 有两个子类实现：ComponentOriginalVisitor 和 HttpSessionProxyVisitor，这里又使用了装饰者模式，Decoratee 是 ComponentOriginalVisitor；而 Decorator 是 HttpSessionProxyVisitor，HttpSessionProxyVisitor 是 HttpSessionBindingListener，也就是说，我们使用了 HttpSession 作为缓存机制，HttpSession 的特点是以用户为 Key 的缓存，符合 s 的缓存机制，当然，我们以后也可以使用更好的缓存机制替换 HttpSession，替换了 HttpSession 根本不必涉及到其他类的更好，因为这里使用模式实现了彻底的解耦。

访问者模式另外一个重要角色：Visitable，它则是那些运行结果需要缓存的组件必须继

承的，注意，这里有一个重要点：不是那些组件本省生成需要缓存，而是它的运行结果需要缓存的。继承 Visitable 这些组件事先必须注册在容器中。

目前 Visitable 有两个子类：

* @see com.jdon.bussinessproxy.dyncproxy.ProxyInstanceFactoryVisitable

* @see com.jdon.bussinessproxy.target.TargetServiceFactoryVisitable

前者主要是动态代理的创建，因为 Proxy.newInstance 频繁执行比较耗费性能，第一次创建后，将动态代理实例保存在 httpSession 中，当然每个 Service 对应一个动态代理实例。

TargetServiceFactoryVisitable 主要是为了缓存那些目标服务的实例，目前这个功能没有激活，特殊情况下才推荐激活。

容器的启动

容器启动主要是将配置文件中注册的组件注册到容器中，并启动容器，这涉及到 container 包下 Config 和 Builder 等几个子包。

Config 包主要负责从 container.xml 和 aspect.xml 读取组件；

Builder 包主要负责向 ContainerWrapper 注册这些组件，注册过程是首先从基础组件（container.xml）开始，然后是拦截器组件(aspect.xml)，最后是用户的服务组件(jdonframework.xml)。

容器的生命周期

容器启动后，容器本身实例是放置在 Web 容器的 ServletContext 中。

容器启动并不是在应用系统部署到 Web 容器时就立即启动，而是该应用系统被第一次访问时触发启动，这个行为是由 ContainerSetupScript 的 startup 触发的，而 startup 方法则是由 ServletContainerFinder 的 findContainer 触发。

当应用系统从 Web 容器中销毁或停止，Jdon 框架容器也就此销毁，最好将你的组件中一些对象引用释放，只要继承 Startable，实现 stop 方法即可。

容器的使用

客户端访问组件必需通过容器进行，这个过程分两步：

从 Web 容器中获得框架容器 ContainerWrapper 实例。

从 ContainerWrapper 容器中查询获得组件实例，有两者实例方式：单例和多例。

关于具体使用方式可见前面章节“如何获得 POJO 实例”等。

这个容器使用过程是通过 finder 包下面的类完成，主要是 ServletContainerFinder 类，ComponentKeys 类保存中一些 container.xml 中配置的组件名称，这些组件名称可能需要在框架程序中使用到，这就需要引起注意，这个类中涉及的组件名称不要在 container.xml 中随意改动，当然这个类在以后重整中争取去除。

AOP 包

Jdon AOP 的设计目前功能比较简单，不包括标准 AOP 中的 Mixin 和 Introduction 等功能；AOP 包下有几个子包：Interceptor、joinpoint 和 reflection，分别涉及拦截器、；拦截器切点和反射机制等功能。

bussinessproxy 包

Jdon 框架从架构角度来说，它是一种业务代理，前台表现层通过业务代理层访问业务服务层，使用 AOP 实现业务代理是一种新的趋势，因此本包功能是和 AOP 包功能交互融合的。

bussinessproxy 包中的 config 子包是获取 jdonframework.xml 中的两种服务 EJB 或 POJO 配置，然后生成 meta 子包中 TargetMetaDef 定义。

target 子包封装了关于这两种服务由元定义实现对象创建的工厂功能，服务对象创建透过访问模式使用了 HttpSession 作为 EJB 实例缓存，这个功能主要是为了 EJB 中有态会话 Bean 实现，这样，客户端通过 getService 获得一个有态会话 Bean 时，无需自行考虑保存这个 Bean 引用到 HttpSession 中这一使用细节了。无态会话 bean 从实战角度发现也可以缓存，提高了性能。

Jdon 框架既然是一种业务代理，那么应该服务于各种前台表现层，Jdon 框架业务代理还适合客户端为 Java 客户端情况下的业务服务访问。这部分功能主要是 com.jdon.bussinessproxy.remote 子包实现的。使用方式参考：http://www.jdon.com/product/ejbinvoker.htm

TargetMetaDef

TargetMetaDef 是目标服务的元定义，TargetMetaDef 主要实现有两种：EJBTargetMetaDef 和 POJOTargetMetaDef，分别是 EJB 服务和 POJO 服务实现。

所谓目标服务元定义也就是程序员在 jdonframework.xml 中定义的 services。

Jdon 框架提供目标服务两种形式：目标服务实例创建和目标服务本身。

ServiceFactory

目标服务创建工厂 ServiceFactory 可以根据目标服务元定义创建一个目标服务实例，这也是 WebAppUtul.getService 获得目标服务的原理实现，

com.jdon.controller.service.DefaultServiceFactory 是 ServiceFactory 缺省实现，在 DefaultServiceFactory 中，主要是通过动态代理获得一个服务实例，使用了 DefaultServiceFactory 来实现动态代理对象的缓存。不必每次使用 Proxy.newinstance 第一次执行后，将其结果保存在 HttpSession 中。

Service

目标服务 Service 则是通过方法 Relection 运行目标服务，只要告知 Service 目标服务的类、类的方法、方法参数类型和方法参数值这些定义，除了方法参数值，其余都是字符串定义，这是 Java 的另外一种调用方式。

security 包

Security 包主要应用于 JdonSD（一个组件库）产品的基于容器的安全权限认证。

strutsutil 包

本包封装了 Jdon 框架基于 Struts 的表现层实现，Jdon 框架的 crud 功能和批量分页查询以及树形显示功能都是在该包中实现，还有上传图片等功能（待测试）。

controller 包

controller 包是 Jdon 框架的整体结构包，主要面向框架使用客户端，其中 WebAppUtil 是经常被调用的类。

cache 子包是框架的缓存机制，这个缓存机制主要面向数据类设计，例如 Model 类，功能类是通过 container 实现管理，功能类使用访问者模式实现结合 HttpSession 实现有状态。

Jdon 框架缺省缓存是使用了开源 OfBiz 的一个缓存，如果你希望使用更好的缓存产品。可以通过继承 Cache 或 LRUCache，然后在 container.xml 中替代如下配置即可：

```
<component name="cache" class="com.jdon.controller.cache.LRUCache" >
    <constructor value="cache.xml"/>
</component>
```

cache 子包提供的缓存不但为 Model 服务，还为批量查询的 PageIterator 创建服务，可参见 PageIteratorSolver 类；另外也为上传图片缓存服务，以后可拓展为更多数据性缓存服务。

Config 子包主要是为读取 pojoService 和 ejbService 配置实现 XML 读取功能。

Events 子包包含的是装载数据 Model 载体在各层之间穿梭，类似信使，通常表现层会将 Model 封装到 EventModel 中，传给服务层的服务，服务处理完毕，如果出错将出错信息封装在 EventModel 对象中，这样表现层可从 EventModel 中查知后台处理是否出错，并返回出错信息。

EventModel 的 setErrors 方法是用于服务层封装处理出错信息的方法，可直接将”id.notfound”这样约定出错信息赋值进去，前台如 struts 可根据 id.notfound 到 Application.properties 中查找本地化信息提示。

Model 子包是封装了框架重要的数据类 Model，Model 既是域模型中的 Model，也是一种 DTO，它是在各层之间传送数据的载体，因此 Model 可以允许嵌套 Model，DynamicModel 也许更加适合临时组装一个传送对象。

Model 是框架的 crud 功能重要角色；PageIterator 是框架批量查询的重要角色。

Pool 子包封装了 Apache Pool 功能，主要用于 Pool 拦截器，如果你的 POJO 服务需要池化提升性能，只要让你的 POJO 服务类继承该包下的 Poolable 即可。

Service 子包封装框架两大有关服务的重要功能：创建一个服务 ServiceFactory.create；或直接运行一个服务 Service.execute。

model 包

model 包主要是围绕 Model 的 crud 功能实现展开，这部分是中间层，尽量做到和表现层 strutsutil 包实现解耦，这样如果有新的表现层如 JSF 可直接和本包发生作用，实现 JSF

下的 crud 功能。

ModelManager 是面向表现层操作 Model 的主要客户端类，ModelForm 和 ModelHandler 都是可能需要客户端继承实现的重要类。

Config 子包主要是从 jdonfamework.xml 中获得 models 相关配置，将这些加载到 ModelMapping 对象中。

Handler 子包是延续 config 子包，model 配置中 Modelhandler 部分配置被加载到 HandlerMetaDef 对象中，然后预先实现 class.forName 运行，HandlerObjectFactory 是创建 ModelHandler 实例工厂，一次性生产 20 个，如果不够用，还可以再生产 20 个，因为 ModelHandler 为 struts 的 action 直接调用，而 action 是多线程，所以使用这种池形式可提高性能，这里使用了自制的池功能，原理简单，池性能好，简单性能就好是 Jdon 框架设计一个原则。

Query 子包主要是为批量查询服务，这些类在持久层被调用。

servicelocator 和 Util 包

这两个包都是工具，大部分类从其它开源借鉴过来，正是因为这些经过实践证明运行良好的基础组件，才使得 Jdon 框架的整体基础牢固，感谢他们。

站在巨人的基础上发展是 Jdon 框架设计的另外一个原则。

更详细的描述将在专门 Jdon 框架设计文档中描述。

技术支持注意

无论 Jdon 框架本身还是使用 Jdon 框架开发的任何系统出现问题，都可以在论坛

http://www.jdon.com/jive/forum.jsp?forum=61&thRange=30

发生错误后，请按下面步骤贴出错误：

1. 需要打开 Jboss/server/default/log/server.log

2. 键入搜索" ERROR "，注意 ERROR 两边各有一个空格，找到第一个错误，那是错误起源，贴到论坛中。

我们也提供 Jdon 框架上门培训和咨询，通过 5 天培训让企业用户完全掌握使用 Jdon 框架开发 J2EE 信息系统。联系 banqiao@jdon.com

全文完